Complex & Intelligent Systems (2022) 8:2383–2404
https://doi.org/10.1007/s40747-022-00655-3

ORIGINAL ARTICLE

Convergence and gradient algorithm of a class of neural networks based on the polygonal fuzzy numbers representation

Gang Sun[1] · Mingxin Wang[1] · Xiaoping Li[2]

Received: 5 July 2021 / Accepted: 11 January 2022 / Published online: 2 February 2022

Abstract
As a special case of general fuzzy numbers, the polygonal fuzzy number can describe a fuzzy object by means of an ordered representation of finite real numbers. Different from general fuzzy numbers, the polygonal fuzzy numbers overcome the shortcoming of complex operations based on Zadeh's traditional expansion principle, and can maintain the closeness of arithmetic operation. Hence, it is feasible to use a polygonal fuzzy number to approximate a general fuzzy number. First, an extension theorem of continuous functions on a real compact set is given according to open set construction theorem. Then using Weierstrass approximation theorem and ordered representation of the polygonal fuzzy numbers, the convergence of a single hidden layer feedforward polygonal fuzzy neural network is proved. Secondly, the gradient vector of the approximation error function and the optimization parameter vector of the network are given by using the ordered representation of polygonal fuzzy numbers, and then the gradient descent algorithm is used to train the optimal parameters of the polygonal fuzzy neural network iteratively. Finally, two simulation examples are given to verify the approximation ability of the network. Simulation result shows that the proposed network and the gradient descent algorithm are effective, and the single hidden layer feedforward network have good abilities in learning and generalization.

Keywords n-polygonal fuzzy numbers · Ordered representation · Polygonal neural network · Convergence · Gradient descent algorithm

Introduction

Artificial neural network has the ability of processing nonlinear information adaptively, and it also can overcome the shortcomings of traditional artificial intelligence methods in intuitive pattern, speech recognition, and unstructured information processing. Therefore, neural networks have been successfully applied in fields of expert system, pattern recognition and intelligent control. In fact, most of the previous study on approximation problems of neural networks stays on the existence of networks, and to realize the construction of network, complex algorithm design and program operation are needed. In 1994, Chen [1] first proposed the approximation problem of system identification using neural network. It was proved that integrable functions on compact set can be approximated by the linear composition of continuous functions of one variable, and a method of identifying dynamic system with the neural network was presented. In 2003, Cao and Xu [2] studied the problem of using a single hidden layer neural network to approximate a continuous function by taking the best polynomial approximation as a measure, and methods of the network construction and approximation speed estimation are proposed. Later, Cao and Zhang et al. [3] gave an algorithm of using neural network to approximate a continuous function in a special distance space. In 2008, Xie et al. [4] proved the existence of single hidden layer neural network interpolation under some certain conditions satisfied for activation function, and a calculation method of connection weights and threshold was given simultaneously. In 2009, Xu and Cao et al. [5] studied the approximation error

This work has been supported in part by National Natural Science Foundation of China (Grant no. 61463019), by Natural Science Foundation of Hunan Province (Grant no. 2019JJ40062), and by Research Foundation of Education Bureau of Hunan Province (Grant no. 20C0565).

✉ Xiaoping Li
lxpmath@126.com

1 School of Science, Hunan Institute of Technology, Hengyang 421002, China

2 School of Management, Tianjin Normal University, Tianjin 300387, China

of using this kind of network interpolation to approximate an objective function. These results greatly expand the further research on the construction method and approximation performance of the single hidden layer neural networks.

As early as 1987, Prof. Kosko first proposed the concept of fuzzy neural network by combining fuzzy set with artificial neural network. And then in 1992, Kosko proved that fuzzy systems can approximate real continuous functions on compact sets with arbitrary precision [6,7]. In 1992, Wang and Mendel [8] proved that the Gaussian fuzzy logic system is a uniform approximator. Meanwhile, a meaningful issue of "Can fuzzy neural network be used as a tool to approximate fuzzy function?" was presented in [8]. In 1994, Buckley et al. first studied the problem of using fuzzy neural networks to approximate continuous fuzzy functions [9,10], and then in 1999, aimed at the issue presented in [8], he pointed out that hybrid fuzzy neural networks can constitute universal approximators for fuzzy functions in [11], yet regular fuzzy neural networks do not. In addition, Buckley conjectured that regular fuzzy neural networks have a universal approximation for continuously increasing fuzzy functions. Thereafter, through the systematic research on regular networks, scholars made some important breakthroughs [12–14]. These results have important theoretical value for further research on fuzzy reasoning, fuzzy control and image restoration technology. Unfortunately, arithmetic operations involved in the above results are all based on traditional Zadeh's extension principle, thus these operations are not closed in fuzzy number space. For example, arithmetic operations are not closed even for simple triangular fuzzy numbers and trapezoidal fuzzy numbers. This disadvantage hinders the wide application of fuzzy number theory. Therefore, how to approximately implement the linear arithmetic operation for general fuzzy numbers is a key problem worthy of attention.

In 2002, Liu [15] first proposed the concept of n-symmetric polygonal fuzzy number based on the idea of segmentation, which overcome the difficulty of Zadeh's extension principle-based arithmetic operation. A polygonal fuzzy neural network (PFNN) model was established initially, and arithmetic operation and fuzzy information processing of n-PFNs were given through the representation of finite ordered real numbers. In 2011, Wang and Li [16] gave an improved concept of n-PFN in the case of equidistant segmentation, and the ordered representation of n-PFN was also proposed. And then the linearization operations of some general fuzzy numbers were realized by transforming general fuzzy numbers into the ordered representation of n-PFNs. In 2012, Baez and Moretti et al. proposed another representation for the polygonal fuzzy numbers and extended it to fuzzy sets on $\mathbb{R}^n$. It was proved that fuzzy set family can constitute a complete separable space when given a generalized Hausdorff metric in [17]. In 2014, Wang and Li [18] discussed the universal approximation of a class of PFNN by introducing the concepts of induction operator and K-integral norm. See [19]. In fact, PFNN is a new network system which is based on an artificial neural network and ordered representation of n-PFN. Its main feature is that connection weights and threshold of the network are both n-PFNs. PFNN mainly uses linear operations of n-PFNs to adjust the parameters directly, PFNN has the characteristics of easy implementation and strong approximation ability.

In 2012, He and Wang [20] designed a conjugate gradient algorithm for the PFNN based on the extended the arithmetic operations of PFNs. See [21]. In 2014, Yang and Wang et al. [22] designed a GA-BP hybrid algorithm to optimize parameters of PFNN by combining the genetic algorithm and BP algorithm. In 2016, Li and Li [23] constructed a single input single output PFNN by means of equidistant partition of domain and interpolation function. In 2018, Wang and Suo [24] designed the connection weights and threshold parameters, and proposed the isolation layered algorithm of the multiple input multiple output PFNN. See [25]. In 2021, Wang and Chen [26] further established the neural network model of the T-S fuzzy system by using the ordered representation of n-PFNs, and proposed the TS firefly algorithm of non-homogeneous linear polygonal T-S fuzzy system based on the flight characteristics of fireflies. See [27–30]. These neural networks constructed by the ordered representation of n-PFNs show the superior performance of PFNN from different aspects.

In 2013, Garg and Sharma [31] first proposed a redundancy allocation problem for multi-objective reliability based on particle swarm optimization (PSO), analyzed the performance of the complex repairable industrial system by using the lambda-tau method of the fuzzy confidence interval, and a hybrid PSO-GA for solving constrained optimization problem was put forward. See [32,33]. In 2016, Gaxiola and Melin et al. [34] used genetic algorithm and PSO to optimize the type-2 fuzzy inference system, and applied the optimized type-2 fuzzy inference system to estimate the type -2 fuzzy weight of bacpropagation neural network. See Refs. [35–37]. Later, Agrawal and Pal et al. did a lot of excellent work by using PSO algorithm and generalized type-2 fuzzy set in [38,39]. Especially, Khater and Ding et al. studied the adaptive online learning and multivariable time series analysis for a class of recurrent fuzzy neural networks in [40,41], respectively. In 2019, Hsieh and Jeng [42] utilized locally weighted polynomial regression to propose a single index fuzzy neural network, and used output an activation function and polynomial function to approximate the constructed network. In 2021, Wang and Xiao [43] constructed an interpolation neural network using the step path method and proved that the network has approximation performance. These fuzzy neural networks and their algorithms not only show their advantages in different aspects, but also lay a theoretical foundation for their further wide application.

The main contributions of this paper include two aspects: one is to prove the convergence of single hidden layer feedforward PFNN based on the extension theorem on a compact set, Weierstrass approximation theorem and the ordered representation of n-PFNs; the other is to propose the gradient vector of the approximation error function and the optimization parameter vector of the constructed network through the operation rules of n-PFNs, and utilize the gradient descent algorithm to iteratively train some optimization parameters of PFNN, so as to design an optimization algorithm. These results lay a foundation for the next step to combine with ordinary neural network to show its unique advantage in pattern recognition and information processing. The main innovation is that n-PFNs and its ordered representations are introduced to describe the input and output expressions of a class of fuzzy neural networks, and an optimization algorithm is designed to realize the linearization of this kind of neural networks. This is mainly because the proposed n-PFNs and their operations do not depend on the traditional Zadeh's extension principle. They not only overcome the complexity of traditional fuzzy number operations, but also realize linearization operations. This is undoubtedly the key to the introduction of n-polygonal fuzzy number. Besides, the backpropagation (BP) algorithm can be designed based on the gradient descent method. It is more suitable for the learning algorithm of a multilayer neural network. Generally, its input and output is a nonlinear mapping relationship, and its information processing ability mainly comes from the multiple combinations of simple nonlinear functions. However, the proposed n-PFNs can not only approximate the general fuzzy number with arbitrary accuracy, but also satisfy the linear operation. Therefore, using the ordered representations of n-PFNs as the input and output of a single hidden layer fuzzy neural network is a linear mapping, which makes the designed algorithm easier to realize the multiple replication ability and information processing ability of nonlinear function than the backpropagation algorithm.

Because the general fuzzy number can not simply achieve the linear operations, it can only rely on Zadeh's extension principle to carry out quite complex arithmetic operations, which has always been a key problem obstructing the development and application of fuzzy number theory. However, the proposed n-PFNs is not only the generalization of triangular or trapezoidal fuzzy numbers, but also any fuzzy number can be transformed into an n-PFNs by the number of subdivision n, so as to avoid Zadeh's expansion principle, realize linear operations and maintain the closeness of arithmetic operations. In addition, polygonal fuzzy numbers can be described by finite ordered real numbers (ordered representation), it not only overcomes the complexity of the operation of general fuzzy numbers, but also maintains some excellent properties of the trapezoidal fuzzy numbers, and can approach general fuzzy numbers with arbitrary accuracy. Therefore, they have obvious advantages in fuzzy information processing. This is our main motivation to introduce n-PFNs as a basic tool to adjust parameters of the proposed feedforward PFNN.

The main contents of each section are as follows. In "n-polygonal fuzzy numbers (n-PFNs)", we review some basic concepts of n-PFNs, polygonal fuzzy value function, single layer feedforward neural network, and so on. Some important lemmas and related arithmetic operations are also given in this section. In "Convergence of the neural network", based on the continuation theorem of continuous function and Weierstrass approximation theorem, a single hidden layer feedforward polygonal fuzzy neural network model is established, and the convergence of the network is proved. In "Gradient descent algorithm", a gradient descent algorithm-based parameter vector iteration optimization method is designed to implement the network training. In "Simulation examples", two simulation examples are given to verify the effectiveness and advantages of the proposed approach. In "Conclusion", the main works of this paper are summarized.

n-polygonal fuzzy numbers (*n*-PFNs)

General fuzzy numbers can't simply carry out linear operations, but can only carry out more complex arithmetic operations by Zadeh's extension principle, which hinders the development and application of fuzzy number theory. Hence, it is of positive significance to introduce the concept of polygonal fuzzy number and discuss its extended operations. For the sake of consistency in the expression of the whole paper, $\mathbb{N}$ indicates natural number set, $\mathbb{R}$ is a real number set, $F_0(\mathbb{R})$ indicates the set of all fuzzy numbers on $\mathbb{R}$, where each fuzzy number $A \in F_0(\mathbb{R})$ satisfies (1)–(2): (1) there is $x_0 \in \mathbb{R}$ so that $A(x_0) = 1$; (2) for any $\alpha \in (0, 1]$, the cut set A_α is a bounded closed interval on $\mathbb{R}$.

Basic definition and ordered representation

Definition 1 For a given fuzzy number $A \in F_0(\mathbb{R})$ and $n \in \mathbb{N}$, if the membership function of A has the following form:

$$A(x) = \begin{cases} \frac{1/n}{a_i^1 - a_{i-1}^1}\left(x - a_{i-1}^1\right) + \frac{i-1}{n}, & x \in \left[a_{i-1}^1, a_i^1\right), \\ & i = 1, 2, \ldots, n, \\ 1, & x \in \left[a_n^1, a_n^2\right], \\ -\frac{1/n}{a_{i-1}^2 - a_i^2}\left(x - a_{i-1}^2\right) + \frac{i-1}{n}, & x \in \left(a_i^2, a_{i-1}^2\right], \\ & i = n, n-1, \ldots, 1, \\ 0, & x \in \left(-\infty, a_0^1\right) \text{ or} \\ & x \in \left(a_0^2, +\infty\right), \end{cases} \tag{1}$$

where $a_0^1 \le a_1^1 \le \cdots \le a_n^1 \le a_n^2 \le \cdots \le a_1^2 \le a_0^2$, then A is called an n-polygonal fuzzy num-

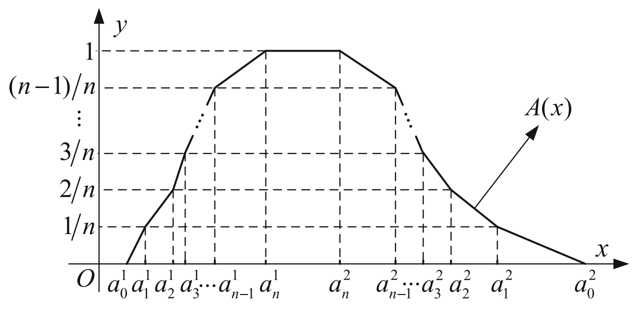

Fig. 1 The membership function image of an n-PFN A

ber on $\mathbb{R}$, and A is also abbreviated as n-PFN, the $2n+2$ ordered real numbers $\{a_0^1, a_1^1, \ldots, a_n^1, a_n^2, \ldots, a_1^2, a_0^2\}$ is called an ordered representation of A, it is expressed as $A = (a_0^1, a_1^1, \ldots, a_n^1, a_n^2, \ldots, a_1^2, a_0^2)$. For a geometric explanation of the membership function $A(x)$, see the following Fig. 1.

According to Eq. (1), it can be known that the membership function of A is continuous, and its image consists of straight line segments. As shown in Fig. 1. It is obvious that the support set and the kernel of A are $\mathrm{Supp}A = (a_0^1, a_0^2)$ and $\mathrm{Ker}A = [a_n^1, a_n^2]$, respectively.

Since the membership function $A(x)$ can be described by $2n+2$ intersections completely, the n-PFN A can be simply represented by the group of intersections. That is $(a_0^1, 0), (a_1^1, \frac{1}{n}), \ldots, (a_n^1, \frac{n}{n}), (a_n^2, \frac{n}{n}), \ldots, (a_1^2, \frac{1}{n}), (a_0^2, 0)$.

Furthermore, since ordinates of these intersections are determined, the n-PFN can be simply represented by the ordered array of abscissas. Thereby, each n-PFN can be uniquely expressed as an ordered representation of $2n+2$ real numbers. On the contrary, the membership function of an n-PFN can also be obtained directly from the ordered representation of the n-PFN in accordance with (1). See the example below.

Example 1 Let an ordered representation $A = \left(-3, -2, 0, \frac{3}{2}, \frac{5}{2}, \frac{10}{3}, 6, 7\right)$, please calculate its corresponding membership function of n-polygonal fuzzy number A (Fig. 2).

In fact, since there are eight real numbers in A, let $2n+2 = 8$ implies $n = 3$, their divided points $\lambda_1 = \frac{1}{3}$, $\lambda_2 = \frac{2}{3}$. It is not difficult to obtain the inflection point coordinates of 3-polygonal fuzzy number A as

$$(-3, 0), \left(-2, \frac{1}{3}\right), \left(0, \frac{2}{3}\right), \left(\frac{3}{2}, 1\right), \left(\frac{5}{2}, 1\right), \left(\frac{10}{3}, \frac{2}{3}\right), \left(6, \frac{1}{3}\right), (7, 0).$$

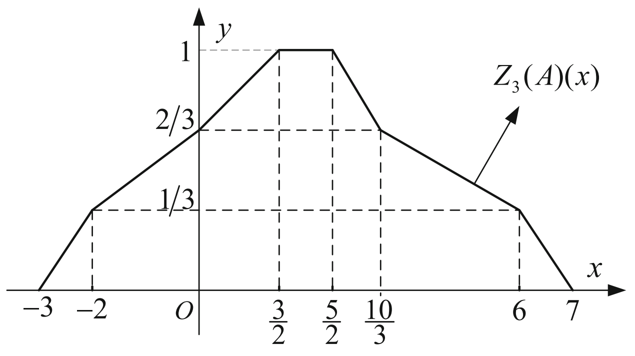

Fig. 2 The membership function image of 3-PFN $Z_3(A)$

Connecting the adjacent inflection points in order with straight line segments, we can get that the membership function and image of A be expressed as

$$A(x) = \begin{cases} \frac{x}{3} + 1, & -3 \le x < -2, \\ \frac{x}{6} + \frac{2}{3}, & -2 \le x < 0, \\ \frac{2x}{9} + \frac{2}{3}, & 0 \le x < \frac{3}{2}, \\ 1, & \frac{3}{2} \le x \le \frac{5}{2}, \\ -\frac{2x}{5} + 2, & \frac{5}{2} < x \le \frac{10}{3}, \\ -\frac{x}{8} + \frac{13}{12}, & \frac{10}{3} < x \le 6, \\ -\frac{x}{3} + \frac{7}{3}, & 6 < x \le 7, \\ 0, & \text{Otherwise}. \end{cases}$$

For ordered representations of n-PFNs, there may be some special cases. For example, if there are $i \in \{1, 2, \ldots, n\}$ and $q \in \{1, 2\}$, and hold $a_{i-1}^q = a_i^q$ for two adjacent intersections, then the corresponding straight line segment of membership function image is vertical. For a more special case of $a_0^1 = a_1^1 = \cdots = a_n^1 = a_n^2 = \cdots = a_1^2 = a_0^2$, A degenerates to a single point fuzzy number, and its membership function image is a vertical line segment. In general, we assume that the inequality $a_0^1 \le a_1^1 \le \cdots \le a_n^1 \le a_n^2 \le \cdots \le a_1^2 \le a_0^2$ holds strictly, that is $a_0^1 < a_1^1 < \cdots < a_n^1 < a_n^2 < \cdots < a_1^2 < a_0^2$. It should be pointed out that this assumption will not affect the subsequent discussion and conclusion.

Let $F_0^{tn}(\mathbb{R})$ be the set of n-PFNs on $\mathbb{R}$. Then it is obvious that $F_0^{tn}(\mathbb{R}) \subset F_0(\mathbb{R})$. In particular, when $n = 1$, the 1-PFN degenerates into a trapezoid fuzzy number or triangle fuzzy number. For an n-PFN ($n \ge 2$), if its membership function image is regarded as the superposition of n small trapezoids or triangle, then the n-PFN can be seen as a generalization of trapezoid fuzzy numbers or triangle fuzzy number.

An important significance of introducing the concept of n-PFNs is that a general fuzzy number can be approximately represented by an n-PFN. In fact, for a general fuzzy number, an n-PFN can be determined according to the value of n. The operation can be described by the following Fig. 3, where

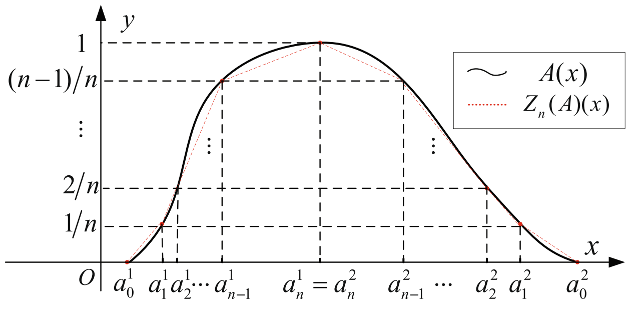

Fig. 3 A general fuzzy number and its approximation of n-PFN

$Z_n(\cdot)$ denotes the membership function of the determined n-PFN. It is not difficult to see from Fig. 3 that the determined n-PFN mainly depends on abscissas of $2n+2$ points on $A(x)$.

It is obvious that the larger the number n is, the more trapezoids or triangle will be obtained, and then the stronger its ability to approximate the general fuzzy number will be. It is of course, with the increase of n, the complexity of the n-PFN increases.

Arithmetic operations and metric

Definition 2 [15,16] For a given $n \in \mathbb{N}$, if $A, B \in F_0^{tn}(\mathbb{R})$, where $A = (a_0^1, a_1^1, \ldots, a_n^1, a_n^2, \ldots, a_1^2, a_0^2)$, $B = (b_0^1, b_1^1, \ldots, b_n^1, b_n^2, \ldots, b_1^2, b_0^2)$, then the corresponding arithmetic operations in $F_0^{tn}(\mathbb{R})$ are defined as follows:

1) $A+B = (a_0^1+b_0^1, a_1^1+b_1^1, \ldots, a_n^1+b_n^1, a_n^2+b_n^2, \ldots, a_1^2+b_1^2, a_0^2+b_0^2)$;
2) $A-B = (a_0^1-b_0^2, a_1^1-b_1^2, \ldots, a_n^1-b_n^2, a_n^2-b_n^1, \ldots, a_1^2-b_1^1, a_0^2-b_0^1)$;
3) $A \cdot B = (c_0^1, c_1^1, \ldots, c_n^1, c_n^2, \ldots, c_1^2, c_0^2)$, where $c_i^1 = a_i^1 b_i^1 \wedge a_i^1 b_i^2 \wedge a_i^2 b_i^1 \wedge a_i^2 b_i^2$, $c_i^2 = a_i^1 b_i^1 \vee a_i^1 b_i^2 \vee a_i^2 b_i^1 \vee a_i^2 b_i^2$, $i = 0, 1, \ldots, n$;
4) $k \cdot A = \begin{cases} (ka_0^1, ka_1^1, \ldots, ka_n^1, ka_n^2, \ldots, ka_1^2, ka_0^2), & k>0, \\ (ka_0^2, ka_1^2, \ldots, ka_n^2, ka_n^1, \ldots, ka_1^1, ka_0^1), & k<0. \end{cases}$

From Definition 2, it can be obtained that compared with general fuzzy number space $F_0(\mathbb{R})$, the arithmetic operations in n-PFN space $F_0^{tn}(\mathbb{R})$ are simple.

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuously increasing activation function, $n \in \mathbb{N}$, we extend the σ as $\sigma : F_0^{tn}(\mathbb{R}) \to F_0^{tn}(\mathbb{R})$. That is to say, if $A = (a_0^1, a_1^1, \ldots, a_n^1, a_n^2, \ldots, a_1^2, a_0^2) \in F_0^{tn}(\mathbb{R})$, we can define $\sigma(A) = (\sigma(a_0^1), \sigma(a_1^1), \ldots, \sigma(a_n^1), \sigma(a_n^2), \ldots, \sigma(a_1^2), \sigma(a_0^2)) \in F_0^{tn}(\mathbb{R})$.

If $n \in \mathbb{N}$, the mapping $Z_n : F_0(\mathbb{R}) \to F_0^{tn}(\mathbb{R})$, then Z_n is called an $n-$polygonal operator. In other words, for any $A \in F_0(\mathbb{R})$, there is an n-PFN $B \in F_0^{tn}(\mathbb{R})$ such that $Z_n(A) = B$. In general, it is easy to obtain the n-PFN $Z_n(A)$ corresponding to $A \in F_0(\mathbb{R})$. See Example 2 below.

Definition 3 [45] Let two given fuzzy numbers $A, B \in F_0(\mathbb{R})$ and $n \in \mathbb{N}$, and their ordered representations are $Z_n(A) = (a_0^1, a_1^1, \ldots, a_n^1, a_n^2, \ldots, a_1^2, a_0^2)$, $Z_n(B) = (b_0^1, b_1^1, \ldots, b_n^1, b_n^2, \ldots, b_1^2, b_0^2)$, define the addition, subtraction and multiplication as follows:

1) $Z_n(A) + Z_n(B) = (a_0^1 + b_0^1, a_1^1 + b_1^1, \ldots, a_n^1 + b_n^1, a_n^2 + b_n^2, \ldots, a_1^2 + b_1^2, a_0^2 + b_0^2)$;
2) $Z_n(A) - Z_n(B) = (a_0^1 - b_0^2, a_1^1 - b_1^2, \ldots, a_n^1 - b_n^2, a_n^2 - b_n^1, \ldots, a_1^2 - b_1^1, a_0^2 - b_0^1)$;
3) $Z_n(A) \cdot Z_n(B) = (c_0^1, c_1^1, \ldots, c_n^1, c_n^2, \ldots, c_1^2, c_0^2)$, where $c_i^1 = a_i^1 b_i^1 \wedge a_i^1 b_i^2 \wedge a_i^2 b_i^1 \wedge a_i^2 b_i^2$ and $c_i^2 = a_i^1 b_i^1 \vee a_i^1 b_i^2 \vee a_i^2 b_i^1 \vee a_i^2 b_i^2$, $i = 0, 1, 2, \ldots, n$;
4) $k \cdot Z_n(A) = (ka_0^1, ka_1^1, \ldots, ka_n^1, ka_n^2, \ldots, ka_1^2, ka_0^2)$, where $k > 0$.

Proposition 1 [16] *Let* $A, B \in F_0(\mathbb{R})$, *for a given* $n \in \mathbb{N}$, *then the following properties hold:*

1) $Z_n(A \pm B) = Z_n(A) \pm Z_n(B)$, $Z_n(A \cdot B) = Z_n(A) \cdot Z_n(B)$;
2) $Z_n(Z_n(A)) = Z_n(A)$, $Z_n(k \cdot A) = k \cdot Z_n(A)$, *where* k *can also be regarded as a set* $\{k\}$.

Example 2 Let the membership functions of the fuzzy numbers A and B be expressed as

$$A(x) = \begin{cases} \sqrt{2x+2} - 1, & -\frac{1}{2} \le x < 1, \\ 1, & 1 \le x \le 2, \\ \sqrt{2 - \frac{x}{2}}, & 2 < x \le 4, \\ 0, & \text{Otherwise;} \end{cases}$$

$$B(x) = \begin{cases} \frac{3x}{x+1}, & 0 \le x < \frac{1}{2}, \\ 1, & \frac{1}{2} \le x \le 1, \\ \frac{2}{x} - 1, & 1 < x \le 2, \\ 0, & \text{Otherwise.} \end{cases}$$

Obviously, Supp$A = [-\frac{1}{2}, 4]$, Ker$A = [1, 2]$; Supp$B = [0, 2]$, Ker$A = [\frac{1}{2}, 1]$.

For example, for the fuzzy number A, if $n = 3$, we choose the divided points $\lambda_1 = 1/3$ and $\lambda_2 = 2/3$. When $x \in [-1/2, 1)$, let $A(x) = \sqrt{2x+2} - 1 = \frac{1}{3}, \frac{2}{3}$, then $x_1 = -\frac{1}{9}$ and $x_2 = \frac{7}{18}$ can be solved, respectively; when $x \in (2, 4]$, let $A(x) = \sqrt{2 - \frac{x}{2}} = \frac{1}{3}, \frac{2}{3}$, then $x_3 = \frac{34}{9}$ and $x_4 = \frac{28}{9}$ can be solved, respectively.

Hence, the ordered representation of 3-polygonal fuzzy number $Z_3(A)$ can be obtained as

$$Z_3(A) = \left(-\frac{1}{2}, -\frac{1}{9}, \frac{7}{18}, 1, 2, \frac{28}{9}, \frac{34}{9}, 4\right).$$

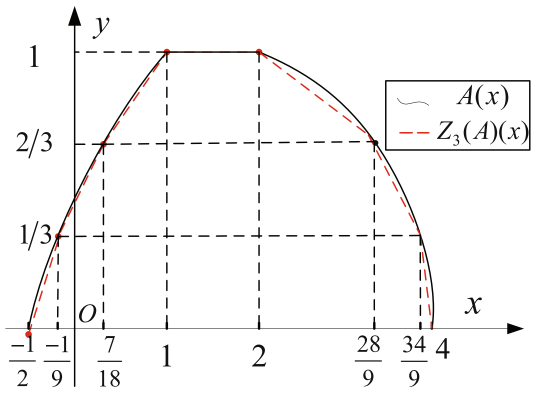

Fig. 4 Images of $A(x)$ and $Z_3(A)(x)$ when $n = 3$

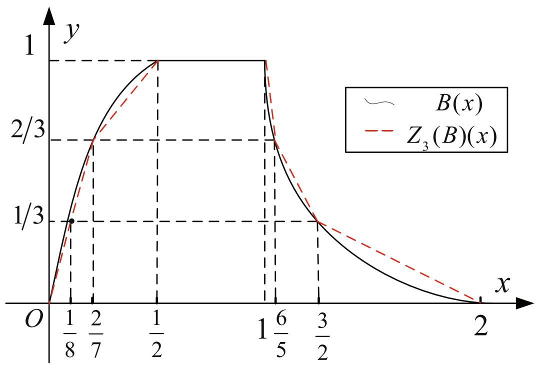

Fig. 5 Images of $B(x)$ and $Z_3(B)(x)$ when $n = 3$

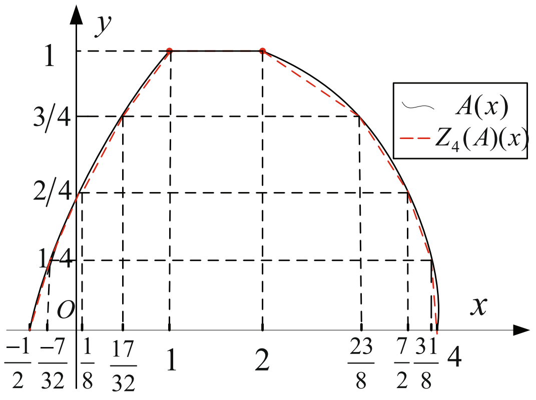

Fig. 6 Images of $A(x)$ and $Z_4(A)(x)$ when $n = 4$

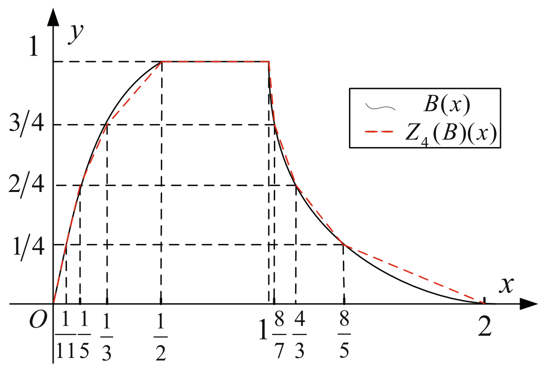

Fig. 7 Images of $B(x)$ and $Z_4(B)(x)$ when $n = 4$

Similarly, if $n = 4$, we choose the divided points $\lambda_1 = \frac{1}{4}$, $\lambda_2 = \frac{2}{4}$ and $\lambda_3 = \frac{3}{4}$, and let $A(x) = \sqrt{2x+2} - 1 = \frac{1}{4}, \frac{2}{4}, \frac{3}{4}$ and $A(x) = \sqrt{2 - \frac{x}{2}} = \frac{1}{4}, \frac{2}{4}, \frac{3}{4}$. It is not difficult to obtain the ordered representation of 4-polygonal fuzzy number $Z_4(A)$ as

$$Z_4(A) = \left(-\frac{1}{2}, -\frac{7}{32}, \frac{1}{8}, \frac{17}{32}, 1, 2, \frac{23}{8}, \frac{7}{2}, \frac{31}{8}, 4\right).$$

Using the same method, we can also easily solve other ordered representations, such as

$$Z_2(A) = \left(-\frac{1}{2}, \frac{1}{8}, 1, 2, \frac{7}{2}, 5\right),$$

$$Z_2(B) = \left(0, \frac{1}{5}, \frac{1}{2}, 1, \frac{4}{3}, 2\right);$$

$$Z_3(B) = \left(0, \frac{1}{8}, \frac{2}{7}, \frac{1}{2}, 1, \frac{6}{5}, \frac{3}{2}, 2\right),$$

$$Z_4(B) = \left(0, \frac{1}{11}, \frac{1}{5}, \frac{1}{3}, \frac{1}{2}, 1, \frac{8}{7}, \frac{4}{3}, \frac{8}{5}, 2\right).$$

According to the membership functions $A(x)$ or $B(x)$ and their ordered representations, we can easily draw their images as follows (Figs. 4, 5, 6, 7):

Clearly, it is not difficult to calculate the analytic expressions of membership function $Z_n(A)(x)$ and $Z_n(B)(x)$. For example, when $n = 3$, we have

$$Z_3(A)(x) = \begin{cases} \frac{6x}{7} + \frac{3}{7}, & -\frac{1}{2} \le x < -\frac{1}{9}, \\ \frac{2x}{3} + \frac{11}{27}, & -\frac{1}{9} \le x < \frac{7}{18}, \\ \frac{6x}{11} + \frac{5}{11}, & \frac{7}{18} \le x < 1, \\ 1, & 1 \le x \le 2, \\ -\frac{3x}{10} + \frac{8}{5}, & 2 < x \le \frac{28}{9}, \\ -\frac{x}{2} + \frac{17}{9}, & \frac{28}{9} < x \le \frac{34}{9}, \\ -\frac{3x}{2} + 6, & \frac{34}{9} < x \le 4, \\ 0, & \text{Otherwise;} \end{cases}$$

$$Z_3(B)(x) = \begin{cases} \frac{8x}{3}, & 0 \le x < \frac{1}{8}, \\ \frac{56x}{27} + \frac{2}{27}, & \frac{1}{8} \le x < \frac{2}{7}, \\ \frac{14x}{9} + \frac{2}{9}, & \frac{2}{7} \le x < \frac{1}{2}, \\ 1, & \frac{1}{2} \le x \le 1, \\ -\frac{5x}{3} + \frac{8}{3}, & 1 < x \le \frac{6}{5}, \\ -\frac{10x}{3} + 2, & \frac{6}{5} < x \le \frac{3}{2}, \\ -\frac{2x}{3} + \frac{4}{3}, & \frac{3}{2} < x \le 2, \\ 0, & \text{Otherwise.} \end{cases}$$

In addition, utilizing Definition 3 and Proposition 1 we can immediately obtain that

$$\begin{cases} Z_3(A+B)=Z_3(A)+Z_3(B) \\ \quad =\left(-\frac{1}{2},\frac{1}{72},\frac{85}{126},\frac{3}{2},3,\frac{194}{45},\frac{95}{18},6\right), \\ Z_3(A-B)=Z_3(A)-Z_3(B) \\ \quad =\left(-\frac{5}{2},-\frac{29}{18},-\frac{73}{90},0,\frac{3}{2},\frac{178}{63},\frac{263}{72},5\right), \\ Z_3(A\cdot B)=Z_3(A)\cdot Z_3(B) \\ \quad =\left(-1,-\frac{1}{6},\frac{1}{9},\frac{1}{2},2,\frac{56}{15},\frac{17}{3},8\right). \end{cases}$$

Similarly, when $n=4$ we have

$$\begin{cases} Z_4(A+B)=\left(-\frac{1}{2},-\frac{45}{352},\frac{13}{40},\frac{83}{96},\frac{3}{2},3,\frac{225}{56},\frac{29}{6},\frac{219}{40},6\right), \\ Z_4(A-B)=\left(-\frac{5}{2},-\frac{291}{160},-\frac{29}{24},-\frac{137}{224},0,\frac{3}{2},\frac{51}{24},\frac{33}{10},\frac{333}{88},4\right), \\ Z_4(A\cdot B)=\left(-1,-\frac{7}{20},\frac{1}{40},\frac{17}{96},\frac{1}{2},2,\frac{23}{7},\frac{14}{3},\frac{31}{5},8\right). \end{cases}$$

In other words, general fuzzy number operations $A \pm B$ and $A \cdot B$ can be approximately expressed as the ordered representations $Z_n(A \pm B)$ and $Z_n(A \cdot B)$, respectively. For example, when $n=3$, we have

$$A+B \approx \left(-\frac{1}{2},\frac{1}{72},\frac{85}{126},\frac{3}{2},3,\frac{194}{45},\frac{95}{18},6\right),$$
$$A\cdot B \approx \left(-1,-\frac{1}{6},\frac{1}{9},\frac{1}{2},2,\frac{56}{15},\frac{17}{3},8\right);$$

when $n=4$,

$$A\cdot B \approx \left(-1,-\frac{7}{20},\frac{1}{40},\frac{17}{96},\frac{1}{2},2,\frac{23}{7},\frac{14}{3},\frac{31}{5},8\right).$$

Remark 1 With the increase of n value, the approximation ability of ordered representation becomes stronger, but its complexity also increases. Therefore, it is very important to choose the appropriate n according to the actual needs. In addition, it is not difficult to see from the above operations that the proposed arithmetic operation does not rely on the traditional Zadeh's extension principle, but only relies on the ordered representation given in Definition 3. In fact, these operations not only overcome the complexity of the traditional fuzzy number expansion operation, but also realize the linearization operation. This is undoubtedly the key to the introduction of n-polygonal fuzzy number. Especially, it has important applications in the approximation theory and optimization algorithm of fuzzy neural network.

In addition, $D(A,B)=\sup_{\alpha\in(0,1]} d_H(A_\alpha,B_\alpha)$ is defined as a distance between two fuzzy numbers in [45], where $A, B \in F_0(\mathbb{R})$, and d_H is a Hausdorff distance. A conclusion is also given that $(F_0(\mathbb{R}), D)$ constitutes a complete metric space. According the definition of general fuzzy number cut set, for any $\alpha \in (0,1]$, let $A_\alpha=\left[a_\alpha^1,a_\alpha^2\right]$, $B_\alpha=\left[b_\alpha^1,b_\alpha^2\right]$, then the distance of fuzzy numbers can be further described as

$$D(A,B)=\sup_{\alpha\in(0,1]}\left(\left|a_\alpha^1-b_\alpha^1\right| \vee \left|a_\alpha^2-b_\alpha^2\right|\right). \tag{2}$$

Lemma 1 [15] *For a given $n\in\mathbb{N}$, if $A, B \in F_0^{tn}(\mathbb{R})$, where $A=\left(a_0^1,a_1^1,\ldots,a_n^1,a_n^2,\ldots,a_1^2,a_0^2\right)$, $B=\left(b_0^1,b_1^1,\ldots,b_n^1,b_n^2,\ldots,b_1^2,b_0^2\right)$, then the distance of n-polygonal fuzzy numbers can be reduced to*

$$D(A,B)=\sup_{0\le i\le n}\left(\left|a_i^1-b_i^1\right| \vee \left|a_i^2-b_i^2\right|\right),$$

and $A \subset B$ if and only if $b_i^1 \le a_i^1 \le a_i^2 \le b_i^2$, $i=0,1,\ldots,n$.

According to Refs. [15,16], it is not difficult to obtain $Z_n(A)(x) \to A(x)$ $(n \to \infty)$ if and only if $\lim_{n\to\infty} D\left(Z_n(A),A\right)=0$.

Remark 2 It is obvious that the arithmetic operations defined in space $F_0^{tn}(\mathbb{R})$ do not depend on Zadeh's extension principle. More importantly, these arithmetic operations are closed and satisfy properties of linearization operations. This not only overcomes the shortcoming of Zadeh's extension principle-based arithmetic operations, but also makes the related operations easy and intuitive. This is the key point of introducing the concept of n-PFNs.

Convergence of the neural network

Modeling of single hidden layer neural network

In this work, a single hidden layer feedforward neural network by using an ordered representation and arithmetic operation of n-PFNs will be established. To do this, the topology of the single hidden layer feedforward neural network model is given firstly in the following figure (Fig. 8).

In the network, the input and output neurons are linear, and activation function of hidden layer neurons is nonlinear. In Fig. 8, the X denotes input signal, O denotes output signal, p denotes the total number of neurons in the hidden layer, U_j and V_j are connection weights, and Θ_j are thresholds of activation function $\sigma(\cdot)$ of the hidden layer neurons, $j=1,2,\ldots,p$. Then the input output expression of the single hidden layer feedforward neural network has the following form,

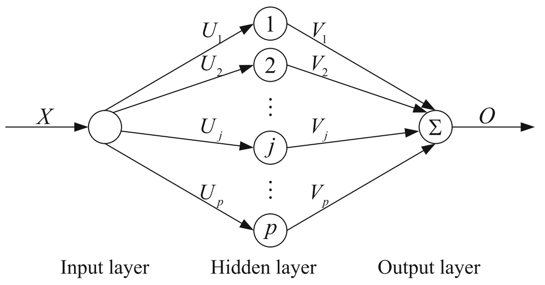

Fig. 8 Topological structure diagram of single hidden layer feedforward neural network

$$O(X) = \sum_{j=1}^{p} V_j \cdot \sigma\left(U_j \cdot X + \Theta_j\right), \tag{3}$$

where $\sigma(\cdot)$ is the activation function of hidden layer neurons.

Remark 3 For a given continuous function f on compact set $K \subset \mathbb{R}$, define the norm of f as $\|f\| = \sup_{x\in K}|f(x)|$. Let $E_n(f) = \inf_{P\in\Delta_n}\|f - P\|$, where Δ_n denotes the set of one variable polynomials of degree n or less, then $E_n(f)$ is called as the best polynomial approximation of f. For the network (3), the following conclusion holds.

Lemma 2 [3] *Let $f \in C(K)$, where $C(K)$ is the set of continuous functions on compact set K. If function $\sigma(\cdot)$ has $n+1$-order continuous derivative in $\mathbb{R}$, and there is a real number $x_0 \in \mathbb{R}$ satisfies $\sigma^{(k)}(x_0) \neq 0$, $k = 0, 1, \ldots, n$, then there must be a single hidden layer feedforward neural network $O(x) = \sum_{j=0}^{p} V_j \cdot \sigma\left(U_j \cdot x + \Theta_j\right)$ holds $\|O - f\| \leq 2E_n(f)$, where the number of neurons in hidden layer of the network is $n+1$.*

Lemma 3 (Weierstrass approximation theorem) *Let $f \in C[a, b]$. Then, for any $\varepsilon > 0$, there is a Bernstein polynomial of degree n holds $|B_n(x) - f(x)| < \varepsilon$, for arbitrary $x \in [a, b]$, where the Bernstein polynomial has the form as $B_n(x) = \sum_{k=0}^{n} f(\frac{k}{n}) C_n^k x^k (1-x)^{n-k}$.*

Definition 4 A mapping $F : D \to F_0^{tn}(\mathbb{R})$ is called an n-polygonal fuzzy valued function, if $F(x) = (f_0^1(x), f_1^1(x), \ldots, f_n^1(x), f_n^2(x), \ldots, f_1^2(x), f_0^2(x)) \in F_0^{tn}(\mathbb{R})$, for arbitrary $x \in D \subset \mathbb{R}$, where $f_i^q : D \to \mathbb{R}$ are continuous functions, $i = 0, 1, \ldots, n$; $q = 1, 2$. A mapping $F : F_0^{tn}(\mathbb{R}) \to F_0^{tn}(\mathbb{R})$ is called a generalized n-polygonal fuzzy valued function, if

$$F(X) = \Big(f_0^1(X), f_1^1(X), \ldots, f_n^1(X), f_n^2(X), \ldots, f_1^2(X), f_0^2(X)\Big) \in F_0^{tn}(\mathbb{R}),$$

for any $X \in F_0^{tn}(\mathbb{R})$.

Hence, according to Definition 4 an n-PFN can be obtained from a real number or an n-PFN by special mapping.

The main objective of this section is to establish an n-polygonal fuzzy numbers based neural network to approximate an n-polygonal fuzzy valued function or generalized n-polygonal fuzzy valued function. After training, the network output can be used instead of an unknown function output, and then the reconstruction problem of unknown functions can be solved.

Convergence of polygonal fuzzy neural network (PFNN)

In this subsection, the PFNN based single hidden layer feedforward neural network model will be established to approximate a continuous n-polygonal fuzzy valued function or generalized n-polygonal fuzzy valued function. In the network, connection weights U_j, V_j and threshold Θ_j are all n-PFNs, and the activation function of the hidden layer neurons $\sigma(\cdot)$ is continuous and monotonically increasing. Thus this class of networks are called as polygonal fuzzy neural networks. According to the operation characteristics of n-PFNs, the PFNN not only has stronger learning ability and fuzzy information processing ability but also are easy to construct and implement. The topology structure of the PFNN is consistent with Fig. 8.

It should be pointed out that the input of PFNN can be real number or n-PFN. Without losing generality, we only consider the case of network input is real number in the following discussion. When the network input is n-PFN, corresponding conclusion can be deduced similarly.

For the convenience of discussion, we assume that the ordered representation of connection weights and threshold are as follows,

$$\begin{cases} U_j = \left(u_0^1(j), u_1^1(j), \ldots, u_n^1(j), u_n^2(j), \ldots, u_1^2(j), u_0^2(j)\right), \\ V_j = \left(v_0^1(j), v_1^1(j), \ldots, v_n^1(j), v_n^2(j), \ldots, v_1^2(j), v_0^2(j)\right), \\ \Theta_j = \left(\theta_0^1(j), \theta_1^1(j), \ldots, \theta_n^1(j), \theta_n^2(j), \ldots, \theta_1^2(j), \theta_0^2(j)\right), \\ j = 1, 2, \ldots, p. \end{cases} \tag{4}$$

Related operations involved in Eq. 3 are subject to Definition 2. In fact, the structure of PFNN is an operation system of addition and multiplication of n-PFNs, and fuzzy information is processed by ordered real numbers.

Definition 5 Let $Q \subset \mathbb{R}$, $F : Q \to F_0^{tn}(\mathbb{R})$ be a continuous n-polygonal fuzzy valued function, and $K \subset Q$ is a compact set. For arbitrary $\varepsilon > 0$, there is $p \in \mathbb{N}$ and $U_j, V_j, \Theta_j \in F_0^{tn}(\mathbb{R})$, $j = 1, 2, \ldots, p$, such that $D(F(x), O(x)) < \varepsilon$, for any $x \in K$, where $O(x)$ is the output of a PFNN of the form Eq. (3), then $O(x)$ is said to have approximation to function $F(x)$.

Lemma 4 [15] *Let $F(x)$ be a continuous n-polygonal fuzzy valued function on Q, and $Q \subset \mathbb{R}$. Then $F(x)$ can be expressed in the form of Eq. (3) if and only if*

$$F(x) = \left(F_0^1(x), F_1^1(x), \ldots, F_n^1(x), F_n^2(x), \ldots, F_1^2(x), F_0^2(x)\right), \quad \text{for any } x \in Q,$$

where $F_i^q(x) = \sum_{j=1}^p v_i^q(j) \cdot \sigma\left(u_i^q(j) \cdot x + \theta_i^q(j)\right)$, $u_i^q(j)$, $v_i^q(j)$ and $\theta_i^q(j)$ are adjustable parameters, $i = 0, 1, \ldots, n$; $q = 1, 2$.

If the adjustment parameters $u_i^q(j)$, $v_i^q(j)$ and $\theta_i^q(j)$ satisfy the conditions (1)-(4) of Theorem 1 given in [15], by letting $h_i^q(j)(x) = v_i^q(j) \cdot \sigma\left(u_i^q(j) \cdot x + \theta_i^q(j)\right)$, then $F_i^q(x)$ can be simplified as $F_i^q(x) = \sum_{j=1}^p h_i^q(j)(x)$, and $h_i^q(j)(x)$ satisfy $h_i^1(j)(x) \le h_{i+1}^1(j)(x) \le h_{i+1}^2(j)(x) \le h_i^2(j)(x)$, $i = 0, 1, \ldots, n-1$; $j = 1, 2, \ldots, p$.

Theorem 1 *Let $f(x)$ be a continuous function on compact set K ($K \subset \mathbb{R}$). Then there must be a closed interval $[a, b] \subset \mathbb{R}$ and a continuous function $\hat{f}(x)$ on $[a, b]$ hold $K \subset [a, b]$ and $f(x) = \hat{f}(x)$, for arbitrary $x \in K$.*

Proof Because K is a compact set in $\mathbb{R}$, then K is bounded. Thus there must be a closed interval $[a, b]$ holds $K \subset [a, b]$.

On the other hand, K is a bounded closed set in $\mathbb{R}$, then $\mathbb{R} - K$ is an open set. According to construction theorem of open set, there must be a family of disjoint open intervals $\{(a_n, b_n)\}$ $(n = 1, 2, \ldots)$ satisfy $\mathbb{R} - K = \bigcup_{n=1}^{+\infty} (a_n, b_n)$.

According to extension theorem of continuous function on closed set, a continuous function $f(x)$ on compact set K can be extended to a continuous function $\hat{f}(x)$ on $\mathbb{R}$. And the function $\hat{f}(x)$ can be constructed as the following form,

$$\hat{f}(x) = \begin{cases} f(x), & x \in K, \\ f(a_n) + \frac{f(b_n) - f(a_n)}{(b_n - a_n)} \cdot (x - a_n), & x \in (a_n, b_n), \\ f(a_n), & x \in (-\infty, a_n), \\ f(b_n), & x \in (b_n, +\infty). \end{cases}$$

It is obvious that the constructed function $\hat{f}(x)$ is continuous at endpoints a_n and b_n of each interval. That is, the extended function $\hat{f}(x)$ is continuous on $\mathbb{R}$, and satisfies $\hat{f}(x) = f(x)$, for arbitrary $x \in K$. □

Theorem 2 *Let $F(x)$ be a continuous n-polygonal fuzzy valued function on compact set K, $K \subset \mathbb{R}$. If monotonically increasing function $\sigma(x)$ has $n+1$-order continuous derivative on $\mathbb{R}$, and there is a point $x_0 \in \mathbb{R}$ satisfies $\sigma^{(k)}(x_0) \neq 0$, $k = 0, 1, \ldots, n$. Then, for any $\varepsilon > 0$, there is a single hidden layer feedforward PFNN of the form (3), such that $D(F(x), O(x)) < \varepsilon$, for any $x \in K$, where $O(x) = \left(o_0^1(x), o_1^1(x), \ldots, o_n^1(x), o_n^2(x), \ldots, o_1^2(x), o_0^2(x)\right)$, $o_i^q(x) = \sum_{j=1}^p v_i^q(j) \cdot \sigma\left(u_i^q(j) \cdot x + \theta_i^q(j)\right)$, where $i = 0, 1, \ldots, n$; $q = 1, 2$.*

Proof According to the known conditions and Definition 4, it can be obtained that the continuous n-polygonal fuzzy value function $F(x)$ can be expressed as

$$F(x) = \left(f_0^1(x), f_1^1(x), \ldots, f_n^1(x), f_n^2(x), \ldots, f_1^2(x), f_0^2(x)\right) \in F_0^{tn}(\mathbb{R}),$$

for any $x \in K$,

where each $f_i^q(x)$ are continuous functions on K, $i = 0, 1, \ldots, n$; $q = 1, 2$, and satisfies

$$f_i^1(x) \le f_{i+1}^1(x) \le f_{i+1}^2(x) \le f_i^2(x), \quad i = 0, 1, 2, \cdots, n-1.$$

According to Theorem 1, for each continuous function $f_i^q(x)$ on $K \subset \mathbb{R}$, there must be a closed interval $\left[a_i^q, b_i^q\right]$ and a continuous function $\hat{f}_i^q(x)$ on $\left[a_i^q, b_i^q\right]$, which satisfy $K \subset \left[a_i^q, b_i^q\right]$ and $\hat{f}_i^q(x) = f_i^q(x)$, for any $x \in K$, $i = 0, 1, \ldots, n$; $q = 1, 2$.

As $K \subset \left[a_i^q, b_i^q\right]$, $i = 0, 1, \ldots, n$; $q = 1, 2$, then $K \subset \bigcap_{i=0}^n \left[a_i^q, b_i^q\right]$, and $\bigcap_{i=0}^n \left[a_i^q, b_i^q\right]$ is a nonempty closed interval. Let $[a, b] = \bigcap_{i=0}^n \left[a_i^q, b_i^q\right]$, clearly, $K \subset [a, b]$. Since each $\hat{f}_i^q(x)$ is continuous on the compact set $[a, b]$, and the activation function $\sigma(\cdot)$ has $n+1$-order continuous derivative. Form Lemma 2, there must be a single hidden layer feedforward neural network o_i^q as

$$o_i^q(x) = \sum_{j=1}^p v_i^q(j) \cdot \sigma\left(u_i^q(j) \cdot x + \theta_i^q(j)\right),$$

for any $x \in [a, b]$,

such that

$$\left\| \hat{f}_i^q - o_i^q \right\| \le 2E_n(\hat{f}_i^q), \; i = 0, 1, \ldots, n; \; q = 1, 2. \tag{5}$$

where $v_i^q(j)$ and $u_i^q(j)$ are connection weights and $\theta_i^q(j)$ is threshold.

By Lemma 3, each continuous function $\hat{f}_i^q(x)$ on $[a, b]$, for arbitrary $\varepsilon > 0$, there must be a Bernstein polynomial $B_n(\hat{f}_i^q, x)$ satisfies

$$\left|\hat{f}_i^q(x) - B_n(\hat{f}_i^q, x)\right| = \left|\hat{f}_i^q(x) - \sum_{k=0}^n \hat{f}_i^q\left(\tfrac{k}{n}\right) C_n^k x^k (1-x)^{n-k}\right| < \varepsilon, \text{ for all } x \in [a, b].$$

According to Formula (5), the best polynomial approximation of each expansion function $\hat{f}_i^q(x)$ on $[a, b]$ must satisfy

$$E_n(\hat{f}_i^q) = \inf_{P\in\Delta_n} \left\| \hat{f}_i^q - P \right\| \le \left\| \hat{f}_i^q - B_n(\hat{f}_i^q) \right\|$$
$$= \sup_{x\in[a,b]} \left| \hat{f}_i^q(x) - B_n(\hat{f}_i^q, x) \right| < \varepsilon.$$

Therefore, for all $x \in [a, b]$, it can be obtained that

$$\left\| \hat{f}_i^q - o_i^q \right\| < 2\varepsilon,\ i = 0, 1, \ldots, n;\ q = 1, 2.$$

And then according to the compact set $K \subset [a, b]$, for any $x \in K$, there must be

$$\left| f_i^q(x) - o_i^q(x) \right| = \left| \hat{f}_i^q(x) - o_i^q(x) \right| \le \left| \hat{f}_i^q - o_i^q \right| < 2\varepsilon. \tag{6}$$

That is to say,

$$f_i^q(x) - 2\varepsilon < o_i^q(x) < f_i^q(x) + 2\varepsilon,$$
$$i = 0, 1, \ldots, n;\ q = 1, 2.$$

In particular, when $q = 1, 2$, it is obvious that

$$f_i^1(x) - 2\varepsilon < \underline{o_i^1(x) < f_i^1(x) + 2\varepsilon},$$
$$\underline{f_{i+1}^1(x) - 2\varepsilon < o_{i+1}^1(x)} < f_{i+1}^1(x) + 2\varepsilon;$$
$$\underline{f_i^2(x) - 2\varepsilon < o_i^2(x)} < f_i^2(x) + 2\varepsilon,$$
$$f_{i+1}^2(x) - 2\varepsilon < \underline{o_{i+1}^2(x) < f_{i+1}^2(x) + 2\varepsilon}.$$

According to the above line drawing inequality, is can be easily got that

$$o_{i+1}^1(x) - o_i^1(x) > f_{i+1}^1(x) - f_i^1(x) - 4\varepsilon,$$
$$o_{i+1}^2(x) - o_i^2(x) < f_{i+1}^2(x) - f_i^2(x) + 4\varepsilon.$$

Form the arbitrariness of ε, for any $x \in K$, we immediately obtain that

$$o_{i+1}^1(x) - o_i^1(x) \ge f_{i+1}^1(x) - f_i^1(x) \ge 0,$$
$$o_{i+1}^2(x) - o_i^2(x) \le f_{i+1}^2(x) - f_i^2(x) \le 0.$$

Therefore, we can also obtained that

$$o_i^1(x) \le o_{i+1}^1(x) \le o_{i+1}^2(x) \le o_i^2(x),$$
$$i = 0, 1, 2, \ldots, n-1.$$

Let

$$O(x) = \left(o_0^1(x), o_1^1(x), \ldots, o_n^1(x), o_n^2(x), \ldots, o_1^2(x), o_0^2(x)\right),$$

then $O(x)$ is an ordered representation of n-polygonal fuzzy value function. In other words, there is a single hidden layer feedforward PFNN in the form of Eq. (3), where

$$o_i^q(x) = \sum_{j=1}^{p} v_i^q(j)\cdot\sigma\left(u_i^q(j)\cdot x + \theta_i^q(j)\right),$$
$$i = 0, 1, \ldots, n;\ q = 1, 2.$$

By Ref. [15], the parameters $u_i^q(j)$, $v_i^q(j)$ and $\theta_i^q(j)$ of each function $o_i^q(x)$ $(i = 0, 1, \ldots, n;\ q = 1, 2)$ can be adjusted appropriately to satisfy the conditions (1)-(4) of Theorem 1 in [15]. If necessary, enough supplementary terms can be added [15,17,23].

According to Lemma 1 and Formula (6), it is not difficult to obtain that

$$D(F(x), O(x)) = \sup_{0\le i\le n} \left(\left| f_i^1(x) - o_i^1(x) \right| \vee \left| f_i^2(x) - o_i^2(x) \right| \right)$$
$$< \sup_{0\le i\le n} (2\varepsilon \vee 2\varepsilon) = 2\varepsilon.$$

Therefore, a single hidden layer feedforward fuzzy neural network (3) can approach the continuous n-polygonal fuzzy value function $F(x)$.

So far, with the help of Weierstrass approximation theorem, the convergence of a single hidden layer feedforward PFNN is proved under the conditions of an activation function satisfies certain conditions, which the network can approximate to a continuous n-polygonal fuzzy valued function is explained. This provides a theoretical basis for further application of the network to mathematical modeling of continuous fuzzy systems. □

Remark 4 In this subsection, an approach for a PFNN to approximate a continuous n-polygonal fuzzy valued function is presented. For a continuous generalized n-polygonal fuzzy valued function, a PFNN can be constructed in the same way. The related operations involved are completely consistent, and only need to expand the network input from real number to n-PFN. This is also the main purpose of this paper to establish a single hidden layer neural network based on the ordered representation of n-polygonal fuzzy numbers.

Gradient descent algorithm

Although the structure of feedforward fuzzy neural network is simple, it shows many advantages in dealing with uncertain problems. It can not only approximate any continuous function and square integrable function with arbitrary accuracy, but also accurately realize any finite training sample set, it is a kind of static nonlinear mapping. However, the feedforward network also has some problems to be solved urgently, such as the optimization of network structure, the design of learning algorithm, the improvement of convergence speed and error accuracy. One of the main reasons for these defects is that the adjustment parameters (fuzzy numbers) of feedforward fuzzy neural network does not meet the linear operation, resulting in that most feedforward networks are learning networks, Due to the lack of certain dynamic behavior, its classification ability and pattern recognition ability are generally weak. In addition, BP algorithm is a local search optimization method. Its weight is gradually adjusted along the direction of local improvement, which usually makes the algorithm fall into local extremum, and the weight converges to a local minimum, and even leads to network training failure, such as many training times, low learning efficiency, slow convergence speed, etc. Therefore, it is an urgent problem to find a basic tool that can better describe fuzzy information and meet linear operations to expand the feedforward fuzzy neural network and backpropagation (BP) network.

At present, no one uses the ordered representation of n-PFNs as a tool to study the approximation of PFNN. Although n-PFN is a special case of general fuzzy number, it can describe fuzzy information with the help of the ordered representation of finite real numbers. Its biggest advantage is that its arithmetic operations avoid the traditional Zadeh's extension principle and approximately realizes the linearization operations. This is the main motivation for us to introduce the n-PFNs and its ordered representation. Especially, it is of great theoretical significance to utilize the n-PFNs to deal with the input and output of fuzzy information.

However, it is more important to realize the nonlinear operation between the general fuzzy numbers involved in the network. The solution of this problem is not only of great significance to realize the optimization algorithm of the neural network, but also provides a theoretical basis for the soft computing technology and application of the network. In fact, the polygonal fuzzy neural network is a new network which depends on the combination of n-polygonal fuzzy number and artificial neural network. It does not need to implement algorithm through general fuzzy number cut set, but design algorithm using the ordered representation of n-PFNs based linear operations, which can greatly simplify the process of designing and optimizing learning algorithm.

In this subsection, the feedforward PFNN with single hidden layer is studied. In the network, input variable is a real number $x \in \mathbb{R}$ or an n-PFN $X = (x_0^1, x_1^1, \dots, x_n^1, x_n^2, \dots, x_1^2, x_0^2) \in F_0^{tn}(\mathbb{R})$. Input output expression of the single hidden layer feedforward PFNN is as follows:

$$O(x) = \sum_{j=1}^{p} V_j \cdot \sigma\left(U_j \cdot x + \Theta_j\right), \tag{7}$$

or

$$O(X) = \sum_{j=1}^{p} V_j \cdot \sigma\left(U_j \cdot X + \Theta_j\right), \tag{8}$$

where the ordered representation of the connection weights and threshold are exactly the same as Eq. (4). The activation function σ of hidden layer neurons is monotonically increasing and differentiable everywhere.

To calculate the derivative of error function conveniently, a metric D_E is introduced to describe the distance of n-PFNs. That is, for any $A, B \in F_0^{tn}(\mathbb{R})$, and $A = \left(a_0^1, a_1^1, \dots, a_n^1, a_n^2, \dots, a_1^2, a_0^2\right)$ and $B = \left(b_0^1, b_1^1, \dots, b_n^1, b_n^2, \dots, b_1^2, b_0^2\right)$, let

$$D_E(A, B) = \sqrt{\sum_{i=0}^{n}\left(\left(a_i^1 - b_i^1\right)^2 + \left(a_i^2 - b_i^2\right)^2\right)}.$$

Obviously, it is not difficult to verify that D_E is a metric (distance). For the sake of unity in discussion, it is assumed that inputs and outputs of the neural network are all n-PFNs, that is the input space and the output space are all $F_0^{tn}(\mathbb{R})$. It needs to be pointed out that the following discussion is valid when the input space is $\mathbb{R}$. In fact, for a real number $x \in \mathbb{R}$, if define $X = (x, x, \dots, x, x, \dots, x, x) \in F_0^{tn}(\mathbb{R})$, then $x \in \mathbb{R}$ can be seen as a special case of $X \in F_0^{tn}(\mathbb{R})$. Thus in the following, we only take the input $X \in F_0^{tn}(\mathbb{R})$ as the representative to discuss.

Let $(X(k); Y(k))$ be k groups of n-PFNs pattern pairs for neural network (8) training, $k = 1, 2, \dots, K$, where $X(k), Y(k) \in F_0^{tn}(\mathbb{R})$, and $X(k)$ is the input of the k-th pattern pair of the network, and $Y(k)$ is the expected output corresponding to $X(k)$. The output of the network corresponding to input $X(k)$ is denoted by $O(k)$, that is the conditions $O(k) = O\left(X(k)\right)$ are satisfied for the network, $k = 1, 2, \dots, K$. For the convenience of discussion, the network inputs, expected outputs and network output are shown in detail as follows:

$$\begin{cases} X(k) = \left(x_0^1(k), x_1^1(k), \dots, x_n^1(k), x_n^2(k), \dots, x_1^2(k), x_0^2(k)\right), \\ Y(k) = \left(y_0^1(k), y_1^1(k), \dots, y_n^1(k), y_n^2(k), \dots, y_1^2(k), y_0^2(k)\right), \\ O(k) = \left(o_0^1(k),\ o_1^1(k), \dots, o_n^1(k), o_n^2(k), \dots, o_1^2(k),\ o_0^2(k)\right), \\ k = 1, 2, \dots, K. \end{cases}$$

For a single hidden layer feedforward PFNN $O(X)$, the error function is defined as

$$\begin{aligned} E\left(O(X)\right) &= \frac{1}{2K}\sum_{k=1}^{K} D_E(O(k), Y(k))^2 \\ &= \frac{1}{2K}\sum_{k=1}^{K}\left(\sum_{i=0}^{n}\left(\left(o_i^1(k) - y_i^1(k)\right)^2 + \left(o_i^2(k) - y_i^2(k)\right)^2\right)\right). \end{aligned} \tag{9}$$

In fact, the structural expression of PFNN shown in Eqs. (7) and (8) are operation systems of the addition and multiplication of n-PFNs. According to the definition of ordered representation, each n-PFN is uniquely determined by $2n+2$ ordered real numbers. Hence, the connection weights U_j, V_j, and threshold Θ_j can be adjusted continuously to make the network output close to the expected output. For the unity of expression, the adjustable parameters $u_i^q(j)$, $v_i^q(j)$ and $\theta_i^q(j)$ $(i = 0, 1, \ldots, n;\ j = 1, 2, \ldots, p;\ q = 1, 2)$ of the networks (7) and (8) are integrated into a high dimensional parameter vector, that is,

$$\begin{aligned} W = \big(&u_0^1(1), u_1^1(1), \ldots, u_n^1(1), u_n^2(1), \ldots, u_1^2(1), u_0^2(1), \ldots,\\ &u_0^1(p), u_1^1(p), \ldots, u_n^1(p), u_n^2(p), \ldots, u_1^2(p), u_0^2(p),\\ &v_0^1(1), v_1^1(1), \ldots, v_n^1(1), v_n^2(1), \ldots, v_1^2(1), v_0^2(1), \ldots,\\ &v_0^1(p), v_1^1(p), \ldots, v_n^1(p), v_n^2(p), \ldots, v_1^2(p), v_0^2(p),\\ &\theta_0^1(1), \theta_1^1(1), \ldots, \theta_n^1(1), \theta_n^2(1), \ldots, \theta_1^2(1), \theta_0^2(1), \ldots,\\ &\theta_0^1(p), \theta_1^1(p), \ldots, \theta_n^1(p), \theta_n^2(p), \ldots, \theta_1^2(p), \theta_0^2(p)\big)\\ \triangleq\ &(w_1, w_2, \ldots, w_i, \ldots, w_N), \end{aligned}$$

where $N = 3p(2n+2)$. Therefore, the error function W defined in Eq. (8) can be simply expressed as $E(W)$ or $E(w_1, w_2, \ldots, w_N)$.

Lemma 5 [15] *Let $E(W)$ be an error function defined by Eq. (8), then the $E(W)$ is differentiable almost everywhere in the high dimensional space $\mathbb{R}^N$, and its partial derivatives $\partial E(W)/\partial w_i$ exist, $i = 1, 2, \ldots, N$.*

For convenience, denote the gradient vector of error function $E(W)$ as $\nabla E(W)$, that is

$$\nabla E(W) = \left(\frac{\partial E(W)}{\partial w_1}, \frac{\partial E(W)}{\partial w_2}, \ldots, \frac{\partial E(W)}{\partial w_N}\right).$$

In the following, a gradient descent algorithm is used to adjust the parameter vector of PFNN. The algorithm program of n-polygonal fuzzy numbers based single hidden layer feedforward neural network is as follows.

Step 1 Determine the initial value of relevant parameters. Let $t = 0$, $\varepsilon > 0$, and $W(t) = W_0$, where t is the iteration number, ε is the iteration termination constant, and W_0 is the initial value of neural network parameter vector. The number of neurons in the network is p.

Step 2 Calculate the neural network training error $E(W(t))$. If the Euclidean norm of $E(W(t))$ satisfies $\|E(W(t))\| < \varepsilon$, the iteration turns to Step 6, Otherwise goes to the next step.

Step 3 Update the neural network parameter vector as $W(t+1) = W(t) - \eta \cdot \nabla E(W(t))$, where η is iteration step size, and $\nabla E(W(t))$ is the gradient vector of $E(W(t))$.

Step 4 Calculate the neural network connection weights $U_j(t+1)$, $V_j(t+1)$ and threshold $\Theta_j(t+1)$, $j \in \{1, 2, \ldots, p\}$. If $U_j(t+1)$, $V_j(t+1)$ and $\Theta_j(t+1)$ do not belong to $F_0^{tn}(\mathbb{R})$, then by adjusting the order of components to make them n-PFNs.

Step 5 Calculate the Euclidean norm of $E(W(t+1))$. If the norm satisfies $\|E(W(t+1))\| < \varepsilon$, the iteration goes to Step 6. Otherwise, let $t = t+1$, and turn to Step 3.

Step 6 Output the parameter vector $W(t+1)$.

In the above algorithm, Step 4 and Step 5 are about the design of adjustment parameters. $U_j(t+1)$, $V_j(t+1)$ and $\Theta_j(t+1)$ represent connection weights and threshold of hidden layer neuron j $(j = 1, 2, \ldots, p)$ corresponding to step $t+1$. It should be noted that these parameters must satisfy the following inequalities after adjusting element order,

$$\begin{cases} U_j(t+1):\ u_0^1(j)(t+1) \le \cdots \le u_n^1(j)(t+1)\\ \qquad\qquad \le u_n^2(j)(t+1) \le \cdots \le u_0^2(j)(t+1),\\ V_j(t+1):\ v_0^1(j)(t+1) \le \cdots \le v_n^1(j)(t+1)\\ \qquad\qquad \le v_n^2(j)(t+1) \le \cdots \le v_0^2(j)(t+1),\\ \Theta_j(t+1):\ \theta_0^1(j)(t+1) \le \cdots \le \theta_n^1(j)(t+1)\\ \qquad\qquad \le \theta_n^2(j)(t+1) \le \cdots \le \theta_0^2(j)(t+1). \end{cases}$$

Remark 5 Because only the gradient information is needed, the above-mentioned gradient descent algorithm is easy to implement. Gradient descent algorithm is a common method of neural network training. To improve the efficiency of network training, some other suitable learning algorithms can be chosen, such as the Newton algorithm, quasi-Newton algorithm. In addition, fixed step size is employed in Step 3 of the given iterative algorithm, such as using variable step size can be considered further. The whole work flowchart can be shown in Fig. 9 below.

In the flowchart shown in Fig. 9, it is not difficult to see that the proposed algorithm has a complete working system and operation rule. According to the construction method of the PFNN approximator given in "Gradient descent algorithm", the structure of the network can be determined by choosing neurons and activation function of them. In addition, by the gradient descent algorithm given in this section, the PFNN approximator can be trained iteratively by using training data, so that we can test the generalization ability of the trained network.

Simulation examples

Because the outputs, connection weights and thresholds of the single hidden layer neural network mentioned above are n-polygonal fuzzy numbers, it is more intuitive and simple to design a learning algorithm on $F_0^{tn}(\mathbb{R})$. Hence, two fuzzy reasoning models which input spaces are $\mathbb{R}$ and $F_0^{tn}(\mathbb{R})$ are simulated by using the presented polygonal fuzzy neural networks (PFNNs), respectively. These models can be applied to automatic control of vehicle speed or automatic operation of container crane, etc. Next, we will give two examples in the

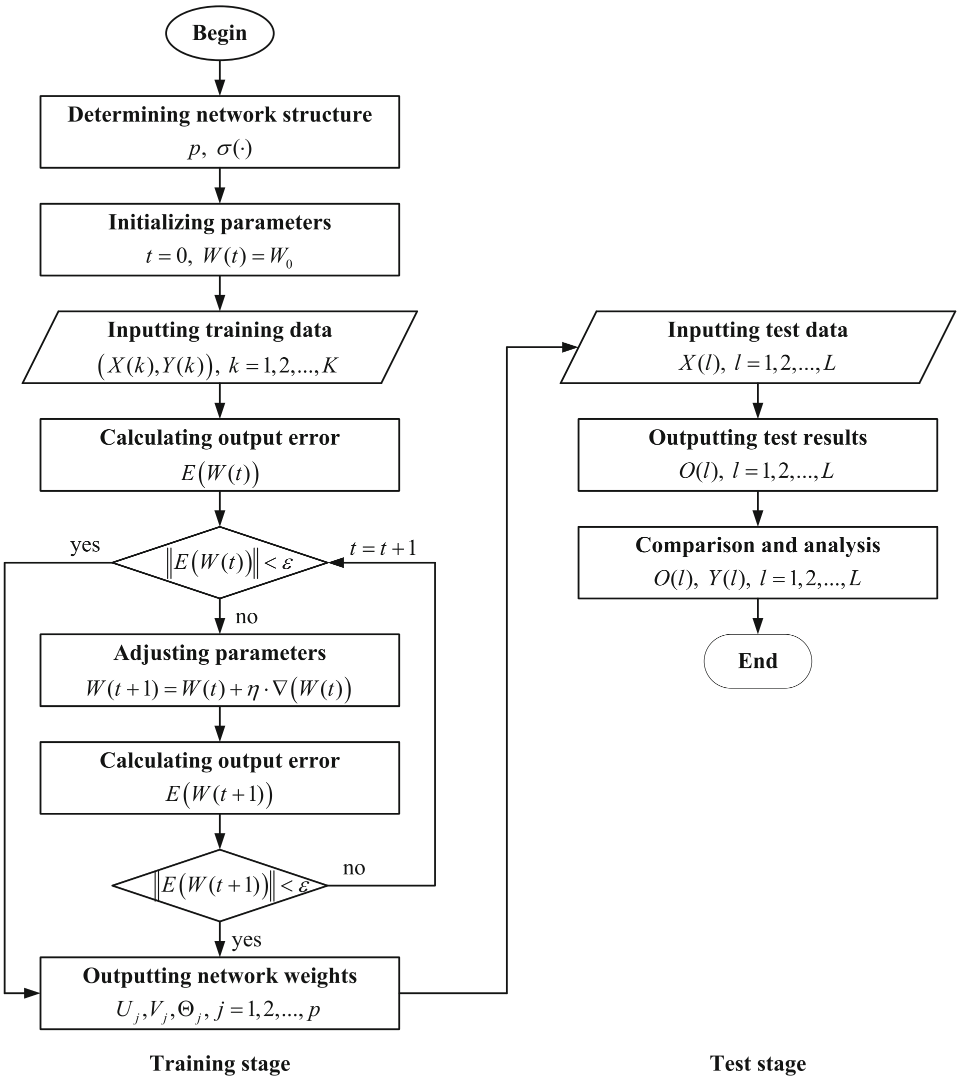

Fig. 9 Work flowchart of training and testing the PFNN approximator

case of $n = 3$ to illustrate the effectiveness of PFNN approximator. To verify the generalization ability of the trained PFNN approximator, we randomly divide the given sample data into two parts: training set and test set, that is, about 70% is used for network training and about 30% for performance testing.

For the following two examples, the actual practice is to randomly take 9 of the 13 samples for training and the remaining 4 for testing. In addition, for the sake of showing advantage and effectiveness of the PFNN approximator by comparison, we construct traditional neural network (TNN) based approximator using the same sample data with PFNN approximator. It should be pointed out that since the dimension of the output space $\mathbb{R}^{2\times3+2} = \mathbb{R}^8$ is eight, the TNN based approximators must be constructed separately to approach each output. Thus, the structure of the TNN approximator is complex. The details are in the following two examples.

Example 3 Input space is $\mathbb{R}$. We want to obtain a PFNN approximator of the form (7). To do this, there are 13 groups of sample pattern pairs $(x(m); Y(m))$ be used in network training and performance testing, where $x(m)$ and $Y(m)$ are inputs and expected outputs of the network, respectively, $m = 1, 2, \ldots, 13$. In fact, these groups of sample pattern pairs come from a 3-polygonal fuzzy valued function $F(x)$, which analytic expression is as follows,

$$F(x) = \left(x^3, x, x^{1/2}, x^{1/4}, (x+1)^{1/4}, (x+1)^{1/2}, (x+1)^{3/2}, (x+1)^2\right), \quad \text{for all } x \in [0,1]. \tag{10}$$

After randomly selecting 13 values of variable x in interval $(0, 1)$, i.e. $x(m)$, $m = 1, 2, \ldots, 13$, then the corresponding 13 values of the 3-polygonal fuzzy valued function $F(x)$ can be obtained according to (10), i.e. $Y(m)$,

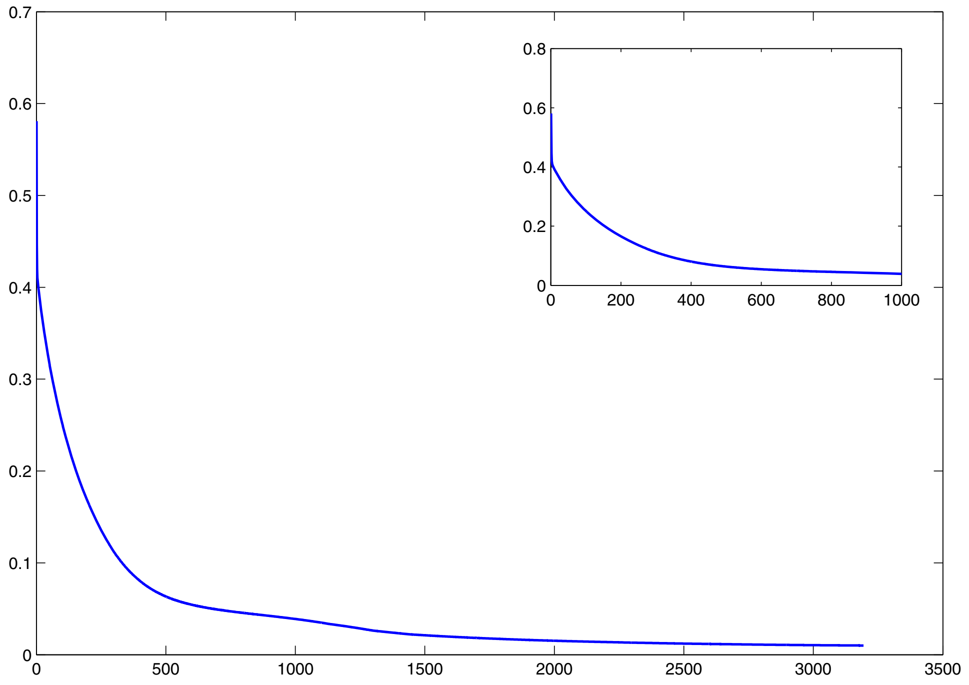

Fig. 10 Training error curve of the PFNN approximator

Table 1 Training inputs $x_{Tr}(k)$, expected outputs $Y_{Tr}(k)$, PFNN outputs $O_{Tr}(k)$, and TNN outputs $O'_{Tr}(k)$

k	$x_{Tr}(k)$	$Y_{Tr}(k)$	$O_{Tr}(k)$	$O'_{Tr}(k)$
1	0.1095	(0.0013, 0.1095, 0.3309, 0.5753, 1.0263, 1.0533, 1.1687, 1.2310)	(−0.0188, 0.2182, 0.4166, 0.6161, 0.9963, 1.0230, 1.1856, 1.2839)	(−0.0586, 0.1918, 0.4419, 0.6867, 1.0724, 1.1418, 1.2262, 1.2724)
2	0.1432	(0.0029, 0.1432, 0.3784, 0.6151, 1.0340, 1.0692, 1.2223, 1.3068)	(−0.0066, 0.2341, 0.4301, 0.6247, 0.9966, 1.0309, 1.2342, 1.3534)	(−0.0384, 0.2180, 0.4617, 0.6971, 1.0717, 1.1478, 1.2786, 1.3455)
3	0.1607	(0.0041, 0.1607, 0.4009, 0.6331, 1.0380, 1.0774, 1.2505, 1.3472)	(0.0001, 0.2428, 0.4375, 0.6296, 0.9972, 1.0354, 1.2600, 1.3911)	(−0.0277, 0.23170.4720, 0.7025, 1.0714, 1.1510, 1.3063, 1.3849)
4	0.3697	(0.0505, 0.3697, 0.6080, 0.7798, 1.0818, 1.1703, 1.6030, 1.8761)	(0.1033, 0.3673, 0.5478, 0.7112, 1.0295, 1.1117, 1.5971, 1.9128)	(0.1113, 0.3958, 0.5954, 0.7659, 1.0670, 1.1883, 1.6493, 1.9216)
5	0.4755	(0.1075, 0.4755, 0.6896, 0.8304, 1.1021, 1.2147, 1.7923, 2.1771)	(0.1723, 0.4461, 0.6197, 0.7695, 1.0644, 1.1666, 1.7864, 2.2212)	(0.1895, 0.4781, 0.6572, 0.7973, 1.0648, 1.2071, 1.8253, 2.2275)
6	0.5246	(0.1444, 0.5246, 0.7243, 0.8511, 1.1112, 1.2348, 1.8826, 2.3245)	(0.2080, 0.4862, 0.6567, 0.8005, 1.0849, 1.1959, 1.8779, 2.3705)	(0.2275, 0.5158, 0.6857, 0.8116, 1.0638, 1.2157, 1.9057, 2.3728)
7	0.7233	(0.3785, 0.7233, 0.8505, 0.9222, 1.1458, 1.3128, 2.2623, 2.9699)	(0.3742, 0.6682, 0.8271, 0.9482, 1.1925, 1.3351, 2.2639, 2.9733)	(0.3903, 0.6632, 0.7983, 0.8682, 1.0596, 1.2504, 2.2129, 2.9457)
8	0.7354	(0.3978, 0.7354, 0.8576, 0.9261, 1.1478, 1.3174, 2.2862, 3.0117)	(0.3852, 0.6802, 0.8383, 0.9581, 1.2002, 1.3445, 2.2879, 3.0082)	(0.4006, 0.6718, 0.8050, 0.8715, 1.0594, 1.2525, 2.2304, 2.9785)
9	0.8472	(0.6081, 0.8472, 0.9204, 0.9594, 1.1658, 1.3591, 2.5106, 3.4122)	(0.4907, 0.7934, 0.9454, 1.0538, 1.2753, 1.4345, 2.5090, 3.3123)	(0.4976, 0.7496, 0.8656, 0.9021, 1.0570, 1.2720, 2.3834, 3.2645)

$m = 1, 2, \ldots, 13$. By grouping them randomly, we can obtain training set and test set, which are expressed as $(x_{Tr}(k); Y_{Tr}(k))$ and $(x_{Te}(l); Y_{Te}(l))$, respectively, $k = 1, 2, \ldots, 9, l = 1, 2, \ldots, 4$.

Before network training, we need to determine the structure of the network. The number of neurons and the activation function of these neurons are chosen as $p = 5$ and $\sigma(x) = \frac{1}{1+e^{-x}}$, respectively. The network training parameters are chosen as $\eta = 0.5$ and $\varepsilon = 0.01$. After doing these, a PFNN approximator can be constructed and trained.

After 3194 iterations, the training process of the PFNN is terminated. Figure 10 shows the change of training error with the increase of iteration times. It can be seen that the training error decreases gradually with the increase of the iteration times.

Remark 6 It should be pointed out that the network training speed is related to the number of neurons p, activation func-

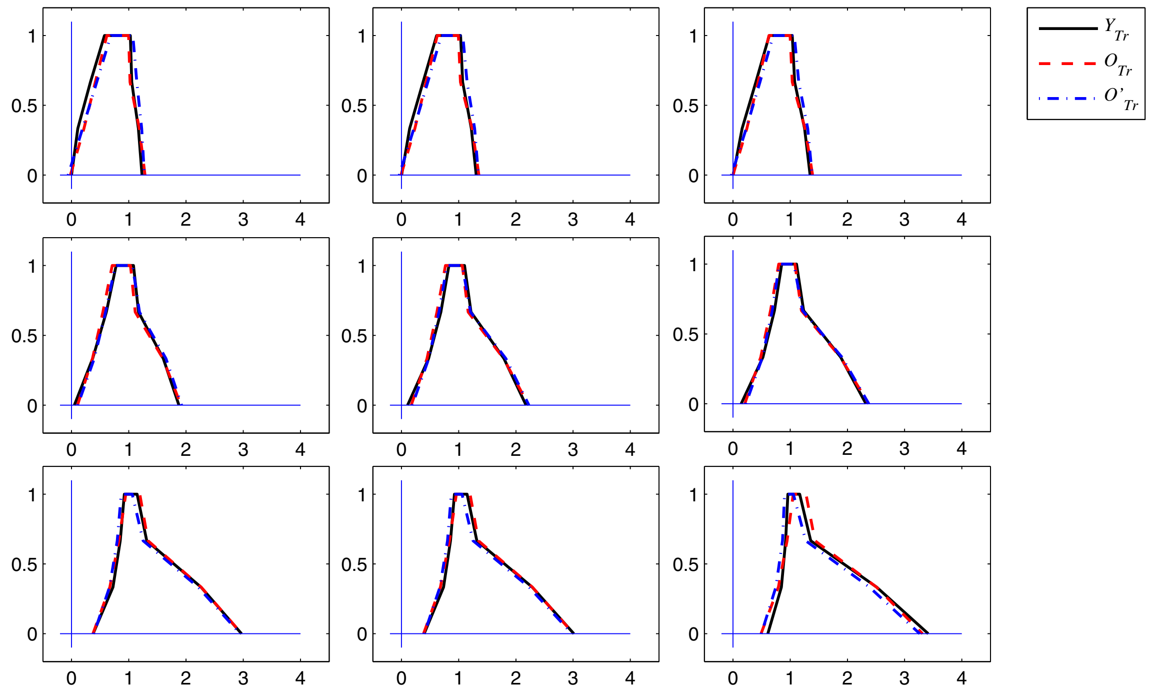

Fig. 11 Training results of the two approximators: PFNN and TNN

Table 2 Test inputs $x_{Te}(l)$, expected outputs $Y_{Te}(l)$, PFNN outputs $O_{Te}(l)$, and TNN outputs $O'_{Te}(l)$

l	$x_{Te}(l)$	$Y_{Te}(l)$	$O_{Te}(l)$	$O'_{Te}(l)$
1	0.2039	(0.0085, 0.2039, 0.4516, 0.6720, 1.0475, 1.0972, 1.3210, 1.4494)	(0.0180, 0.2653, 0.4569, 0.6429, 1.0000, 1.0478, 1.3255, 1.4880)	(−0.0007, 0.2656, 0.4975, 0.7158, 1.0705, 1.1587, 1.3755, 1.4862)
2	0.2297	(0.0121, 0.2297, 0.4793, 0.6923, 1.0531, 1.1089, 1.3637, 1.5123)	(0.0295, 0.2795, 0.4694, 0.6517, 1.0026, 1.0560, 1.3657, 1.5487)	(0.0159, 0.2859, 0.5128, 0.7237, 1.0699, 1.1633, 1.4175, 1.5493)
3	0.3490	(0.0425, 0.3490, 0.5907, 0.7686, 1.0777, 1.1615, 1.5668, 1.8197)	(0.0912, 0.3532, 0.5310, 0.7011, 1.0241, 1.1022, 1.5614, 1.8554)	(0.0966, 0.3795, 0.5832, 0.7597, 1.0675, 1.1846, 1.6142, 1.8637)
4	0.5551	(0.1710, 0.5551, 0.7450, 0.8631, 1.1167, 1.2470, 1.9392, 2.4182)	(0.2313, 0.5121, 0.6807, 0.8209, 1.0989, 1.2151, 1.9355, 2.4640)	(0.2515, 0.5389, 0.7033, 0.8204, 1.0632, 1.2211, 1.9548, 2.4629)

tion $\sigma(\cdot)$ and iteration parameters η, ε, etc. In addition, it can be seen that the training algorithm converges faster in early stage and slower in a later stage. By improving algorithm, the overall convergence efficiency may be improved.

The specific input and expected output data used for the PFNN approximator training is given in Table 1 below (see the second and third columns of the table). The actual output data of the PFNN approximator is shown in the fourth column of Table 1. To give numerical comparison results, outputs of the TNN approximator are shown in the fifth column of Table 1.

In addition, to intuitively show the training performance of the PFNN approximator and the TNN approximator, images of these 3-PFNs are given in Fig. 11.

Clearly, it is not difficult to see from Fig. 11 that the training outputs of the PFNN approximator and the TNN approximator can approach the expected outputs well.

In the following, we will verify the generalization ability of the trained PFNN approximator and compare it with the trained TNN approximators. We input $x_{Te}(l)$ into the trained PFNN approximator, and then the corresponding test outputs $O_{Te}(l)$ can be obtained, $l = 1, 2, \ldots, 4$. See the fourth column of Table 2 for specific data. To compare test results, the expected outputs and the test outputs of the TNN approximator are also given in Table 2, where $Y_{Te}(l)$ is the expected outputs and $O'_{Te}(l)$ the TNN outputs, $l = 1, 2, \ldots, 4$.

In addition, in order to intuitively give the comparison of the test results, images of the outputs, that are $Y_{Te}(l)$, $O_{Te}(l)$, and $O'_{Te}(l)$, $l = 1, 2, \ldots, 4$, are shown in Fig. 12.

We can see from Fig. 12 that the test outputs of the PFNN approximator and the TNN approximator can approach the expected outputs well.

Next, we will numerically analyze distances between outputs of approximators and expected outputs to verify the

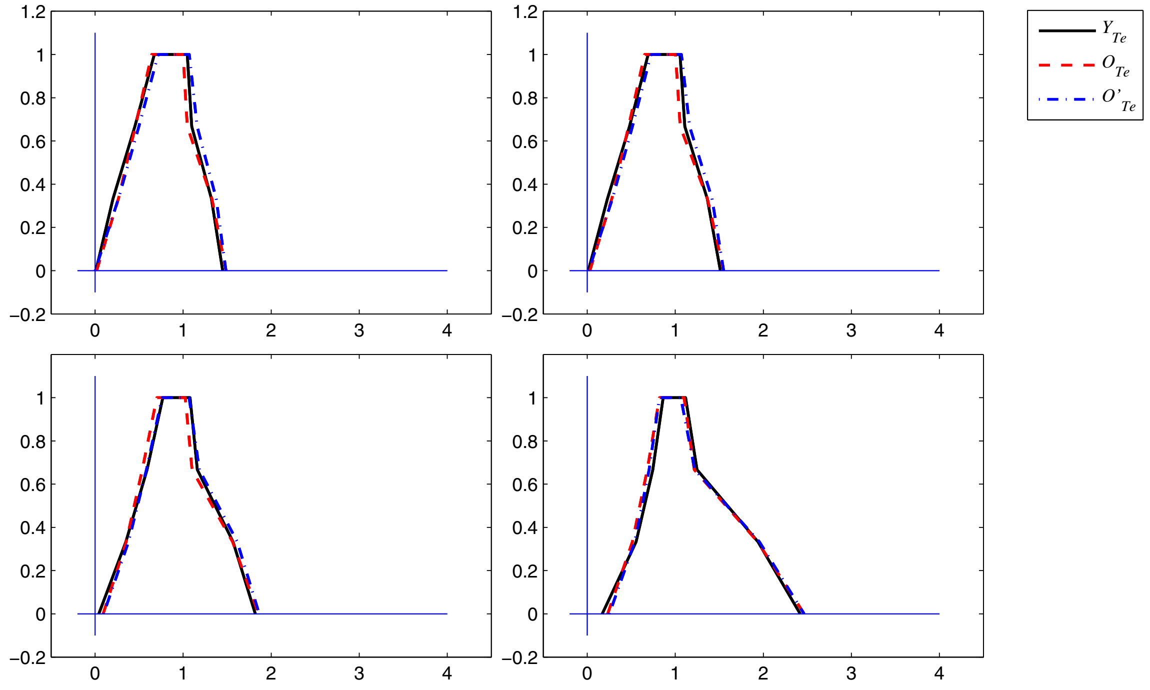

Fig. 12 Test results of the two approximators: PFNN and TNN

performance of the PFNN and TNN. Since these outputs are all 3-PFNs, we use the distance of 3-PFNs given in Lemma 1 for numerical analysis. It is easy to understand that the smaller the distance values, the stronger the approximation ability of the network.

For convenience, let $O(m)$ be outputs of PFNN approximator corresponding to $x(m)$, and $O'(m)$ outputs of the TNN approximator, $m = 1, 2, \ldots, 13$. According to Tables 1 and 2, the distance data shown in Table 3 can be obtained, where $D(Y(m), O(m))$ and $D\left(Y(m), O'(m)\right)$ are the distances in the sense of Lemma 1, $m = 1, 2, \ldots, 13$.

To compare the performance of the PFNN approximator and the TNN approximator, According to Table 3, the scatter plots of these distance data are given in Fig. 13 as follows:

Table 3 Distances between outputs of approximators and expected outputs

m	$D(Y(m), O(m))$	$D\left(Y(m), O'(m)\right)$
1	0.1087	0.1114
2	0.0910	0.0833
3	0.0821	0.0736
4	0.0686	0.0608
5	0.0699	0.0820
6	0.0676	0.0831
7	0.0551	0.0861
8	0.0553	0.0884
9	0.1174	0.1477
10	0.0613	0.0617
11	0.0529	0.0561
12	0.0675	0.0541
13	0.0643	0.0805

Remark 7 It should be pointed out that generally speaking, the approximation effect of the PFNN and the TNN can be further improved by increasing the number of neurons p or decreasing the value of iteration parameter ε, etc.

Example 4 Input space is $F_0^{tn}(\mathbb{R})$. We want to obtain a PFNN approximator of the form (8). To do this, there are 13 groups of sample pattern pairs $(X(m); Y(m))$ be used in network training and performance test, where $X(m)$ and $Y(m)$ are inputs and expected outputs of the network, respectively, $m = 1, 2, \ldots, 13$. In fact, these groups of sample pattern pairs come from a 3-polygonal fuzzy valued function $F : F_0^{t3}(\mathbb{R}) \to F_0^{t3}(\mathbb{R})$, which analytic expression is given as follows. For all $x \in [0, 1]$,

$$X = \Big(x^3, x, x^{\frac{1}{2}}, x^{\frac{1}{4}}, (x+1)^{\frac{1}{4}}, (x+1)^{\frac{1}{2}}, (x+1)^{\frac{3}{2}}, (x+1)^2\Big) \in F_0^{tn}(\mathbb{R}), \tag{11}$$

$$F(X) = \Big(x^4, x^{\frac{3}{2}}, x, x^{\frac{1}{4}}(x+1)^{\frac{1}{4}}, (x+1)^{\frac{1}{2}}, x^2+x+1, (x+1)^2, (x+1)^3\Big). \tag{12}$$

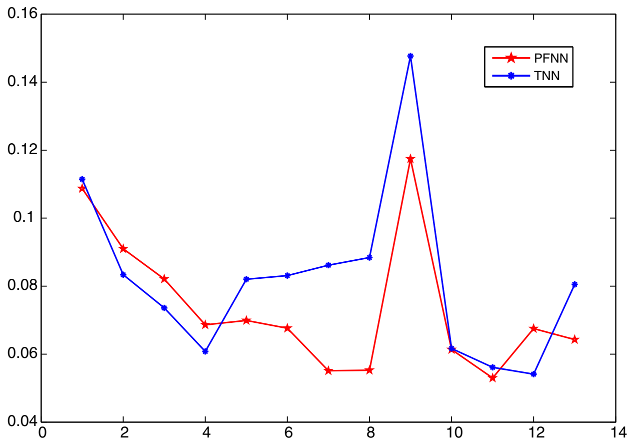

Fig. 13 Scatter plot of distances

After randomly selecting 13 values of variable x in interval (0, 1), then inputs and outputs of sample pattern pairs $X(m)$ and $Y(m)$ can be obtained according to (11) and (12), respectively, $m = 1, 2, \dots, 13$. Then after random grouping, there are 9 groups of sample pattern pairs in training set and 4 groups of sample pattern pairs in test set. Let $(X_{Tr}(k); Y_{Tr}(k))$ represent the training set, and $(X_{Te}(l); Y_{Te}(l))$ the test set, $k = 1, 2, \dots, 9$, $l = 1, 2, \dots, 4$.

We determine the structure of the network which has the form of (8) as follows: the number of neurons is $p = 4$, and activation function of neurons is $\sigma(x) = \frac{1}{1+e^{-x}}$. Parameters used for the PFNN training are chosen as $\eta = 0.2$ and $\varepsilon = 0.1$, respectively. The specific data of input and output used for approximator training is given in Table 4 (See the second and third columns of the table for details).

After 3423 iterations, training process of the PFNN is terminated. Figure 14 shows the change of training error with the increase of iteration times, that is, the training error decreases gradually with the increase of the iteration times.

From Figs. 10 and 14, it can be seen that the training error decreases gradually with the increase of the number of iterations. Especially in the initial stage of iteration, the error decreases very fast. It shows that the gradient descent algorithm given in "Gradient descent algorithm" is effective.

Similar to Example 3, we use the same training set to construct a TNN approximator and compare its performance with the PFNN approximator. The training outputs of the two approximators are also given in Table 4, where $O_{Tr}(k)$ for the PFNN approximator and $O'_{Tr}(k)$ for the TNN approximator, $k = 1, 2, \dots, 9$.

In addition, to intuitively show the training performance of the two approximators, Fig. 15 gives the images of these outputs. We can see that the training outputs of the PFNN approximator and the TNN approximator approach the expected outputs well.

From Figs. 11 and 15, it can be seen that the training outputs of the PFNN approximator are close to the expected outputs. It shows that the proposed methods of PFNN approximator constructing and training in "Convergence of the neural network" and "Gradient descent algorithm" are effective.

Then, we test the generalization ability of the two approximators with the test inputs $X_{Te}(l)$, $l = 1, 2, \dots, 4$, and compare the output results. The test inputs, expected out-

Table 4 Training inputs $X_{Tr}(k)$, expected outputs $Y_{Tr}(k)$, PFNN outputs $O_{Tr}(k)$, and TNN outputs $O'_{Tr}(k)$

k	$X_{Tr}(k)$	$Y_{Tr}(k)$	$O_{Tr}(k)$	$O'_{Tr}(k)$
1	(0.1638, 0.5471, 0.7397, 0.8600, 1.1153, 1.2438, 1.9243, 2.3936)	(0.0896, 0.4047, 0.5471, 0.9592, 1.2438, 1.8464, 2.3936, 3.7031)	(0.1160, 0.4062, 0.5364, 0.9464, 1.2361, 1.6629, 2.4769, 3.7488)	(−0.0203, 0.2827, 0.4123, 0.7979, 1.0893, 1.8497, 2.3895, 3.7358)
2	(0.1473, 0.5281, 0.7267, 0.8525, 1.1118, 1.2362, 1.8890, 2.3350)	(0.0778, 0.3838, 0.5281, 0.9478, 1.2362, 1.8070, 2.3350, 3.5681)	(0.1153, 0.4024, 0.5333, 0.9430, 1.2340, 1.6555, 2.4308, 3.6332)	(−0.0228, 0.2784, 0.4105, 0.7946, 1.0869, 1.8349, 2.3543, 3.6178)
3	(0.1348, 0.5127, 0.7160, 0.8462, 1.1090, 1.2299, 1.8605, 2.2882)	(0.0691, 0.3671, 0.5127, 0.9384, 1.2299, 1.7755, 2.2882, 3.4613)	(0.1149, 0.3994, 0.5307, 0.9402, 1.2323, 1.6496, 2.3941, 3.5420)	(−0.0247, 0.2750, 0.4091, 0.7918, 1.0850, 1.8228, 2.3260, 3.5240)
4	(0.0389, 0.3390, 0.5822, 0.7630, 1.0757, 1.1571, 1.5494, 1.7928)	(0.0132, 0.1973, 0.3390, 0.8208, 1.1571, 1.4538, 1.7928, 2.4005)	(0.1114, 0.3652, 0.4987, 0.9029, 1.2123, 1.5817, 2.0195, 2.6624)	(−0.0428, 0.2420, 0.3903, 0.7593, 1.0627, 1.6824, 2.0184, 2.6056)
5	(0.2599, 0.6382, 0.7989, 0.8938, 1.1313, 1.2799, 2.0968, 2.6837)	(0.1659, 0.5098, 0.6382, 1.0112, 1.2799, 2.0455, 2.6837, 4.3964)	(0.1196, 0.4246, 0.5508, 0.9618, 1.2459, 1.6976, 2.7100, 4.3411)	(−0.0073, 0.3048, 0.4201, 0.8133, 1.1007, 1.9186, 2.5587, 4.3210)
6	(0.4343, 0.7573, 0.8702, 0.9329, 1.1514, 1.3256, 2.3296, 3.0881)	(0.3289, 0.6590, 0.7573, 1.0741, 1.3256, 2.3308, 3.0881, 5.4268)	(0.1262, 0.4490, 0.5682, 0.9796, 1.2581, 1.7425, 3.0449, 5.1800)	(0.0126, 0.3369, 0.4287, 0.8322, 1.1150, 2.0019, 2.7754, 5.0775)
7	(0.3153, 0.6806, 0.8250, 0.9083, 1.1386, 1.2964, 2.1787, 2.8244)	(0.2146, 0.5615, 0.6806, 1.0342, 1.2964, 2.1438, 2.8244, 4.7468)	(0.1216, 0.4333, 0.5572, 0.9684, 1.2503, 1.7137, 2.8254, 4.6346)	(−0.0006, 0.3159, 0.4233, 0.8202, 1.1059, 1.9493, 2.6369, 4.5963)
8	(0.0103, 0.2175, 0.4664, 0.6829, 1.0504, 1.1034, 1.3434, 1.4824)	(0.0022, 0.1015, 0.2175, 0.7174, 1.1034, 1.2649, 1.4824, 1.8049)	(0.1104, 0.3421, 0.4715, 0.8677, 1.1972, 1.5331, 1.7976, 2.2018)	(−0.0517, 0.2262, 0.3744, 0.7334, 1.0470, 1.5829, 1.8234, 2.1383)
9	(0.1201, 0.4933, 0.7024, 0.8381, 1.1055, 1.2220, 1.8249, 2.2301)	(0.0592, 0.3465, 0.4933, 0.9265, 1.2220, 1.7367, 2.2301, 3.3302)	(0.1143, 0.3955, 0.5274, 0.9365, 1.2302, 1.6421, 2.3489, 3.4304)	(−0.0271, 0.2708, 0.4072, 0.7884, 1.0825, 1.8075, 2.2905, 3.4086)

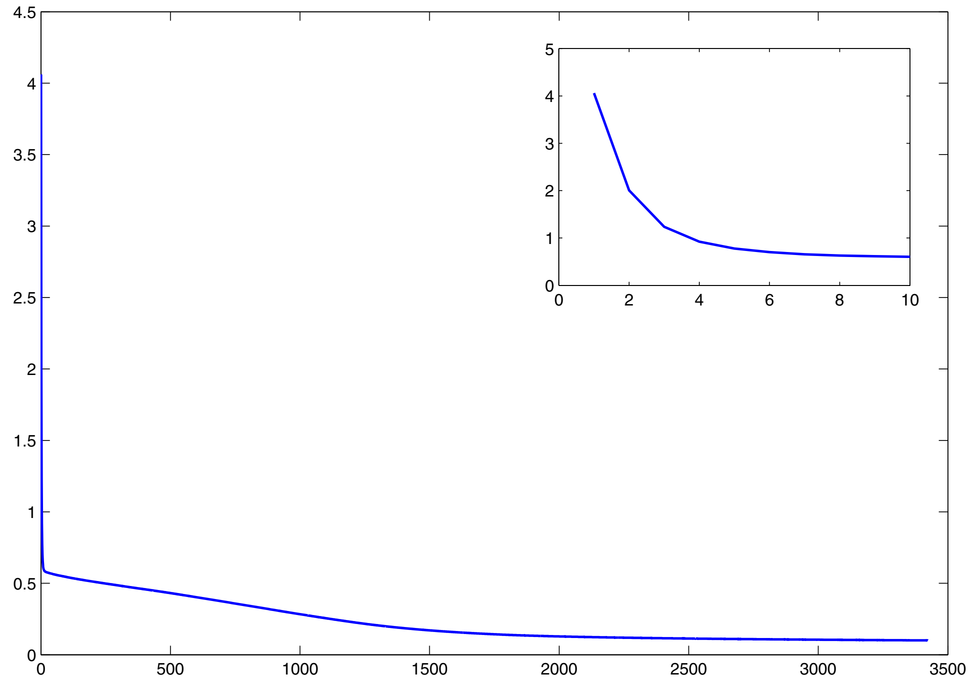

Fig. 14 Training error curve of the PFNN approximator

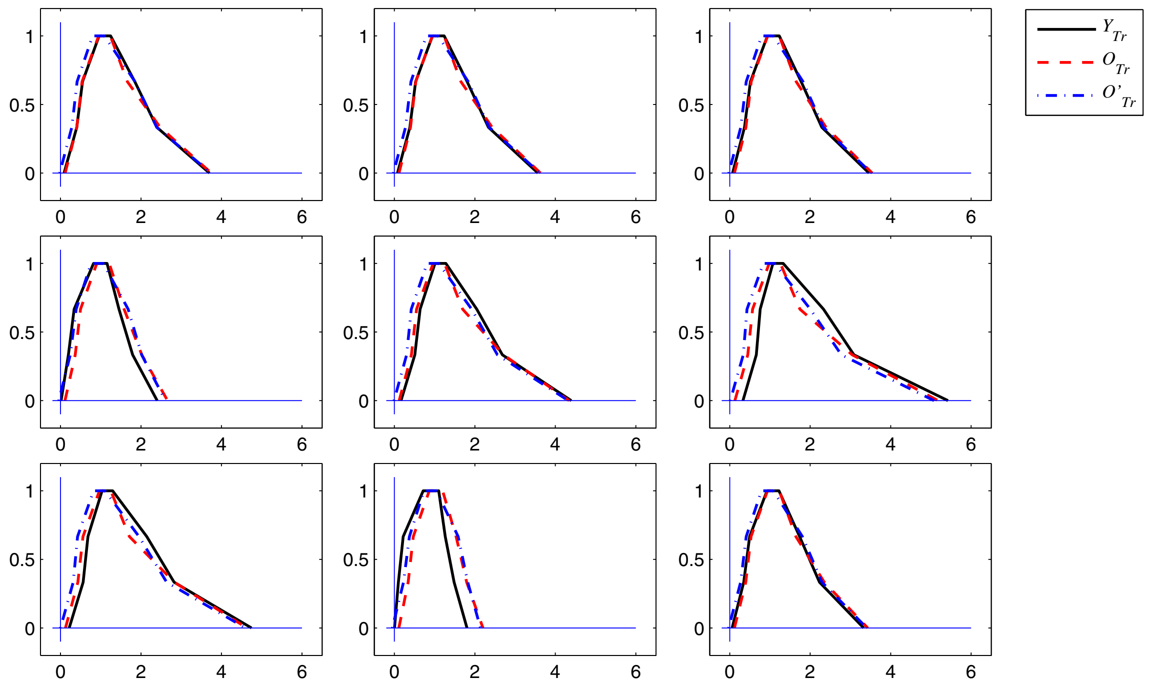

Fig. 15 Training results of the two approximators: PFNN and TNN

puts, and the outputs of the two approximators are given in Table 5, where $Y_{Te}(l)$ are the expected outputs, $O_{Te}(l)$ the outputs of PFNN approximator, and $O'_{Te}(l)$ the outputs of the TNN approximator, $l = 1, 2, \ldots, 4$.

According to Table 5, the test results of the PFNN approximator and the TNN approximator are shown in Fig. 16. We can see that the test outputs of the PFNN approximator and the TNN approximator can approach the expected outputs well.

From Figs. 12 and 16, it can be seen that the test outputs of the PFNN approximator are close to the expected outputs, i.e. that the trained PFNN approximator has good generalization ability. This further shows the effectiveness of the proposed PFNN constructing approach.

Table 5 Test inputs $X_{Te}(l)$, expected outputs $Y_{Te}(l)$, PFNN outputs $O_{Te}(l)$, and TNN outputs $O'_{Te}(l)$

l	$X_{Te}(l)$	$Y_{Te}(l)$	$O_{Te}(l)$	$O'_{Te}(l)$
1	(0.1199, 0.4932, 0.7023, 0.8380, 1.1054, 1.2220, 1.8246, 2.2295)	(0.0592, 0.3463, 0.4932, 0.9264, 1.2220, 1.7364, 2.2295, 3.3291)	(0.1143, 0.3955, 0.5274, 0.9365, 1.2302, 1.6421, 2.3485, 3.4294)	(−0.0271, 0.2707, 0.4072, 0.7884, 1.0825, 1.8073, 2.2902, 3.4076)
2	(0.0401, 0.3424, 0.5851, 0.7649, 1.0764, 1.1586, 1.5553, 1.8020)	(0.0137, 0.2003, 0.3424, 0.8234, 1.1586, 1.4596, 1.8020, 2.4189)	(0.1114, 0.3659, 0.4994, 0.9038, 1.2127, 1.5831, 2.0262, 2.6771)	(−0.0425, 0.2426, 0.3907, 0.7600, 1.0631, 1.6852, 2.0242, 2.6209)
3	(0.1074, 0.4753, 0.6894, 0.8303, 1.1021, 1.2146, 1.7919, 2.1765)	(0.0510, 0.3277, 0.4753, 0.9151, 1.2146, 1.7012, 2.1765, 3.2111)	(0.1139, 0.3919, 0.5243, 0.9330, 1.2282, 1.6352, 2.3075, 3.3295)	(−0.0292, 0.2670, 0.4054, 0.7851, 1.0802, 1.7931, 2.2577, 3.3036)
4	(0.4706, 0.7779, 0.8820, 0.9391, 1.1547, 1.3334, 2.3705, 3.1608)	(0.3661, 0.6860, 0.7779, 1.0844, 1.3334, 2.3829, 3.1608, 5.6194)	(0.1276, 0.4532, 0.5711, 0.9825, 1.2602, 1.7502, 3.1059, 5.3268)	(0.0164, 0.3427, 0.4300, 0.8353, 1.1173, 2.0154, 2.8114, 5.1996)

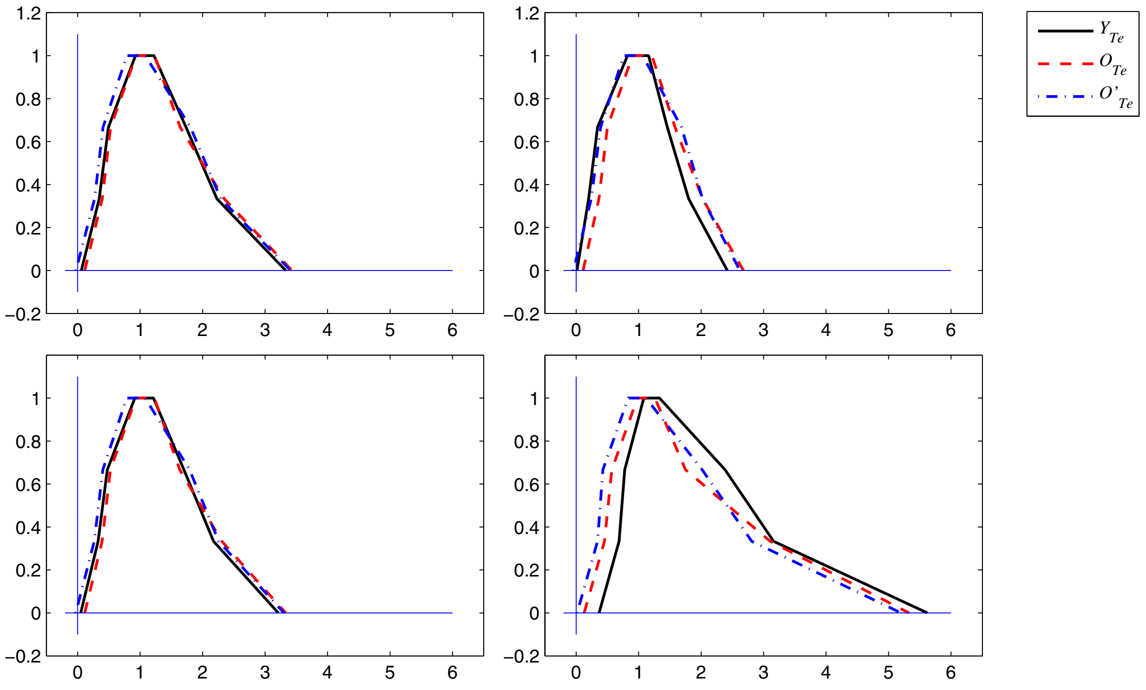

Fig. 16 Test results of the two approximators: PFNN and TNN

Let $O(m)$ be outputs of the PFNN approximator corresponding to the input $X(m)$, and $O'(m)$ the outputs of the TNN approximator, $m = 1, 2, \ldots, 13$. According to Tables 4-5, the distance data can be obtained as shown in Table 6, where $D(Y(m), O(m))$ and $D\left(Y(m), O'(m)\right)$ are the distances in the sense of Lemma 1, $m = 1, 2, \ldots, 13$.

According to Table 6, the scatter plots of the distances $D(Y(m), O(m))$ and $D\left(Y(m), O'(m)\right)$ are given in Fig. 17 below.

From Figs. 13 and 17, we can conclude that after training, the PFNN can achieve the same level of approximation performance with TNN. However, it should be emphasized again that to realize the output of 3-PFNs, the TNN approximator needs eight independent neural networks to output each corresponding component, respectively. So the structure of TNN approximator is more complex. Therefore, in comparison, the PFNN approximator has the advantage of a simple structure.

By analyzing the results of the two simulation examples, we can get that the proposed PFNN approximator construction method and the gradient descent algorithm are effective. From Tables 1, 4 and Figs. 11, 15, it can be known that the proposed method of PFNN approximators constructing and training is effective. Figs. 10 and 14 show that the given gradient descent algorithm has good convergence. In addition, it can be seen from Tables 2, 5 and Figs. 12, 16 that the trained PFNN approximator have good generalization ability. Figs. 11, 12, 15, 16 and Tables 3, 6 indicate that the constructed PFNN approximator has good approximation efficiency. Therefore, the constructed polygonal fuzzy neural network based on ordered representation of n-PFNs has some advantages over the traditional neural network.

Table 6 Distances between outputs of approximators and expected outputs

m	$D\,(Y(m), O(m))$	$D\left(Y(m), O'(m)\right)$
1	0.1835	0.1613
2	0.1515	0.1532
3	0.1259	0.1466
4	0.2619	0.2286
5	0.3479	0.2181
6	0.5883	0.3493
7	0.4301	0.2573
8	0.3969	0.3410
9	0.1188	0.1395
10	0.1190	0.1395
11	0.2582	0.2256
12	0.1310	0.1344
13	0.6327	0.4198

Conclusion

As a universal approximator, a neural network has the ability to learn and approximate any unknown continuous function, while fuzzy theory has a unique effect in dealing with problems with uncertainty. Fortunately, the proposed n-PFNs not only satisfies the linear operation, but also can approximately describe the general fuzzy numbers through the ordered representation of finite real numbers. Therefore, selecting n-PFNs as the adjustment parameter of feedforward fuzzy neural network can make the network form a linear mapping, so as to linearize the input and output of the constructed network. A feedforward neural network with a single hidden layer is established by Weierstrass approximation theorem under the ordered representation of n-polygonal fuzzy number. It is proved that the proposed network has the approximation to a continuous n-polygonal fuzzy value function or a generalized n-polygonal fuzzy value function. This provides a theoretical basis for the application of the fuzzy neural network with single hidden layer in a continuous system. A gradient descendent algorithm is designed by using the iterative operations of parameter vectors in n-PFNs space. The effectiveness of the network and training algorithm is verified by simulation examples. A single layer PFNN is established by the ordered representation of n-PFNs, and the method of describing fuzzy information by n-PFNs can be further extended to the construction of other fuzzy neural networks, such as the convolutional neural network and recurrent fuzzy neural network, and then explore and design some intelligent algorithms of these networks. Besides, to improve the efficiency and convergence speed of network training, the n-PFNs can be further considered to be applied to genetic algorithm or particle swarm optimization algorithm, and then the improvement of gradient descent algorithm and variable step-size method are also the focus of the next step research.

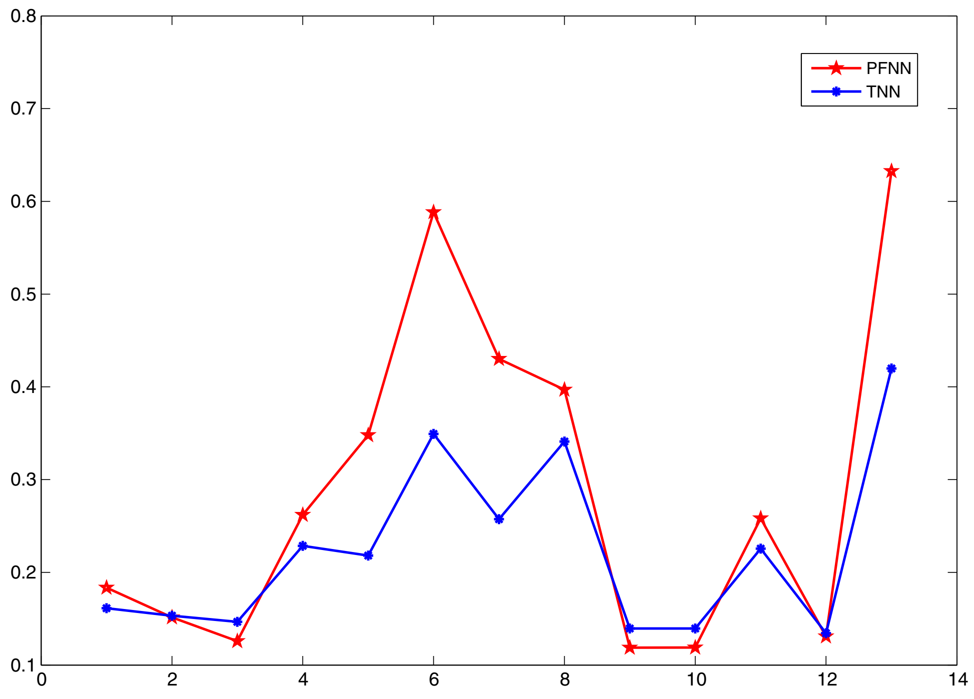

Fig. 17 Scatter plot of distances

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant no. 61463019), by Natural Science Foundation of Hunan Province (Grant no. 2019JJ40062), and by Research Foundation of Education Bureau of Hunan Province (Grant no. 20C0565).

References

1. Chen TP (1994) Neural network and its approximation problem in system identification. Sci China (Series A) 24(1):1–7
2. Cao FL, Xu ZB, Liang JY (2003) Approximation of polynomial functions by neural network: construction of network and algorithm of approximation. Chin J Comput 26(8):906–912
3. Cao FL, Zhang YQ, Zhang WG (2007) Neural networks with single hidden layer and the best polynomial approximation. Acta Math Sinica 50(2):385–392
4. Xie TF, Cao FL (2008) On the construction of interpolating neural networks. Prog Nat Sci 18(3):334–340
5. Xu SY, Cao FL (2009) Estimation of error for interpolation neural networks in distance spaces. J Syst Sci Math Sci 29(5):670–676
6. Kosko B (1992) Neural networks and fuzzy systems: a dynamical systems approach to intelligence. Prentice-Hall, Englewood Cliffs
7. Kosko B (1992) Fuzzy systems as universal approximators. In: Proceedings of IEEE international conference on fuzzy systems, pp 1153–1162
8. Wang LX, Mendel JM (1992) Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. Trans Neural Netw 3(5):807–814
9. Buckley JJ, Hayashi Y (1994) Fuzzy neural networks: a survey. Fuzzy Sets Syst 66:1–13
10. Buckley JJ, Hayashi Y (1994) Can fuzzy neural nets approximate continuous fuzzy functions? Fuzzy Sets Syst 61(1):43–51
11. Buckley JJ, Hayashi Y (1999) Can neural nets be universal approximators for fuzzy functions? Fuzzy Sets Syst 58:273–278
12. Feuring T, Lippe WM (1999) The fuzzy neural network approximation lemma. Fuzzy Sets Syst 102(2):227–237
13. Liu PY (2001) Universal approximation of continuous analyses fuzzy valued functions by multi-layer regular fuzzy neural networks. Fuzzy Sets Syst 119(2):303–311
14. Liu PY, Li HX (2005) Approximation analysis of feedforward regular fuzzy neural network with two hidden layers. Fuzzy Sets Syst 150(2):373–396
15. Liu PY (2002) A new fuzzy neural network and its approximation capability. Sci China (Series E) 32(1):76–86
16. Wang GJ, Li XP (2011) Universal approximation of polygonal fuzzy neural networks in sense of K-integral norms. Sci China Inf Sci 54(11):2307–2323
17. Baez-Sanchez AD, Moretti AC, Rojas-Medar MA (2012) Polygonal fuzzy sets and numbers. Fuzzy Sets Syst 209(1):54–65
18. Wang GJ, Li XP (2014) Construction of the polygonal fuzzy neural network and its approximation based on K-integral norm. Neural Netw World 24(4):357–376
19. Zhao FX, Li HX (2006) Universal approximation of regular fuzzy neural networks to Sugeno-integrable functions. Acta Math Appl Sin 29(1):39–45
20. He Y, Wang GJ (2012) The conjugate gradient algorithm of the polygonal fuzzy neural networks. Acta Electron Sin 40(10):2079–2084
21. Sui XL, Wang GJ (2012) Influence of perturbations of training pattern pairs on stability of polygonal fuzzy neural network. Pattern Recogn Artif Intell 26(6):928–936
22. Yang YQ, Wang GJ, Yang Y (2014) Parameters optimization of polygonal fuzzy neural networks based on GA-BP hybrid algorithm. Int J Mach Learn Cybern 5(5):815–822
23. Li XP, Li D (2016) The structure and realization of a polygonal fuzzy neural network. Int J Mach Learn Cybern 7(3):375–389
24. Wang GJ, Suo CF (2018) The isolation layered optimization algorithm of MIMO polygonal fuzzy neural network. Neural Comput Appl 29(10):721–731
25. Wang GJ, Gao JS (2019) Parallel conjugate gradient-particle swarm optimization and the parameters design based on the polygonal fuzzy neural network. J Intell Fuzzy Syst 37(1):1477–1489
26. Wang GJ, Chen X, Sun G (2021) Design and optimization of TS firefly algorithm based on nonhomogeneous linear polygonal T-S fuzzy system. Int J Intell Syst 36(2):691–714
27. Wang H, Wang W, Zhou X et al (2017) Firefly algorithm with neighborhood attraction. Inf Sci 382:374–387
28. Wang H, Zhou X, Sun H et al (2017) Firefly algorithm with adaptive control parameters. Soft Comput 21(17):5091–5102
29. Li RY, Chen QQ, Chen SY (2017) Dynamic search firefly algorithm based on improved attraction. Pattern Recogn Artif Intell 30(6):538–548
30. He LF, Huang SW (2017) Modified firefly algorithm based multilevel thresholding for color image segmentation. Neurocomputing 240:152–174
31. Garg H, Sharma SP (2013) Multi-objective reliability-redundancy allocation problem using particle swarm optimization. Comput Ind Eng 64(1):247–255
32. Garg H (2013) Performance analysis of complex repairable industrial systems using PSO and fuzzy confidence interval based lambda-tau methodology. ISA Trans 52(2):171–183
33. Garg H (2016) A hybrid PSO-GA algorithm for constrained optimization problems. Appl Math Comput 274:292–305
34. Gaxiola F, Melin P, Valdez F et al (2016) Optimization of type-2 fuzzy weights in backpropagation learning for neural networks using GAs and PSO. Appl Soft Comput 38:860–871
35. Gaxiola F, Melin P, Valdez F et al (2017) Comparison of T-norms and S-onrms for interval type-2 fuzzy numbers in weight adjustment for neural networks. Information 8(114):1–21
36. Gaxiola F, Melin P, Valdez F et al (2015) Generalized type-2 fuzzy weight adjustment for backpropagation neural networks in time series prediction. Inf Sci 325:159–174
37. Gaxiola F, Melin P, Valdez F et al (2019) PSO with dynamic adaptation of parameters for optimization in neural networks with interval type-2 fuzzy numbers weights. Axioms 8(14):1–21
38. Agrawal S, Agrawal J, Kaur S et al (2018) A comparative study of fuzzy PSO and fuzzy SVD-based RBF neural network for multi-label classification. Neural Comput Appl 29:245–256
39. Pal SS, Kar S (2019) A hybridized forecasting method based on weight adjustment of neural network using generalized type-2 fuzzy set. Int J Fuzzy Syst 21:308–320
40. Khater AA, El-Nagar AM, El-Bardini M et al (2020) Online learning based on adaptive learning rate for a class of recurrent fuzzy neural network. Neural Comput Appl 32:8691–8710

41. Ding H, Li W, Qiao J (2021) A self-organizing recurrent fuzzy neural network based on multivariate time series analysis. Neural Comput Appl 33:5089–5109
42. Hsieh JG, Jeng JH, Lin YL et al (2019) Single index fuzzy neural networks using locally weighted polynomial regression. Fuzzy Sets Syst 368:82–100
43. Wang GJ, Xiao WM, Tao YJ (2021) Interpolation neural network constructed by the step path and its approximation performance. Clust Comput 24(2):1397–1411
44. Diamond P, Kloeden P (1994) Metric spaces of fuzzy sets. World Scientific Press, Singapore
45. Wang GJ (2017) Polygonal fuzzy neural network and fuzzy system approximation. Science Press, Beijing